Broadview
www.broadview.com.cn

测试实践丛书

51testing
软件测试网作品系列

畅销书升级版

让复杂变成简单，
让简单成为行动！

性能测试从零开始

——LoadRunner入门与提升

组编　　柳 胜 编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版
Broadview
www.broadview.com.cn

性能测试从零开始——LoadRunner入门与提升

专家推荐

本书理论系统完整，逻辑清楚，丝丝入扣。从性能测试的目的到实现途径，从性能测试流程到实践应用，本书给出一个完整的解决方案。作者根据多年性能测试实践，提出了独创的GAME性能测试模型，对现实工作有非常强的指导意义。

51Testing软件测试培训高级讲师 朴春龙

本书作为LoadRunner的入门教材，其宗旨是将性能测试的要点、LoadRunner术语一一罗列并分析，并通过对LoadRunner的介绍，深入浅出地把这些概念和知识点展现给读者一副清晰的图卷，能使读者达到融会贯通，举一反三的效果。

阿里巴巴(中国)网络技术有限公司技术部 测试专家 梁剑钊

本书以LoadRunner工具为线索，对性能测试理论、技术、策略、方法、实践进行深入浅出的分析和总结，帮助读者建立正确清晰的性能测试思路，获得高效的基于LoadRunner的性能测试能力。

微软全球总部Windows产品部 高级测试工程师 张志刚

本书旨在培养新一代软件性能测试导演，使你具备如下良好的导演素质：

- 具有领导的能力，表现为能够自如掌控LoadRunner。
- 具有全局的视角，表现为能够使用GAME模型迅速规划并开展一场演出，并使其顺利完成，达到最好的效果。
- 具有实干的精神，表现为理论与实践相结合，本书有很多的实际案例供您借鉴参考。

源代码免费下载，网址为：

- http://www.51testing.com/51testing.php
- http://www.broadview.com.cn

柳胜的联系方式：http://www.51testing.com/?uid/2382
个人网站：http://www.cesoo.info/

上架建议：性能测试>LoadRunner

ISBN 978-7-121-12732-8

9 787121 127328 >

定价：50.00元

责任编辑：李 冰
责任美编：李 玲

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。

测试实践丛书

51testing 软件测试网作品系列

性能测试从零开始

——LoadRunner入门与提升

51Testing软件测试网 组编　　柳 胜 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

LoadRunner 是一个内涵丰富、功能强大的性能测试工具，这是许多专业人士选择它的原因。初学者面对许多的性能测试概念和 LoadRunner 专业术语，常常感到无所适从，这是因为 LoadRunner 作为软件性能测试的实现者，涉及了软件测试及软件体系架构众多方面的知识。可以说，学习 LoadRunner 是理解和学习软件性能测试的非常好的切入点。

本书作为 LoadRunner 的入门教材，其宗旨是将性能测试的要点、LoadRunner 术语一一罗列并分析，并通过对 LoadRunner 的介绍，深入浅出地把这些概念和知识点展现给读者。通过学习本书，可以学会使用 LoadRunner 做性能测试。第 1~7 章，奠定性能测试和 LoadRunner 的理论基础，其中第 4 章讲述了 LoadRunner 脚本语言基础。第 8~10 章，通过实际案例来活学活用，总结和提取最佳性能测试实践，并介绍 LoadRunner 的使用技巧。第 11 章是性能量化分析模型，能够指导用户在性能测试过程中从定性到定量分析的转变。第 12 章对当前最前沿的性能测试框架做了一个前瞻分析，为有志于在性能测试领域深造的朋友提供参考思路。

本书可作为刚进入性能测试领域的测试工程师的入门参考书，对已经有一定工作经验的朋友来说，也是一本整理思路、融会贯通的总结笔记。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

性能测试从零开始：LoadRunner 入门与提升 / 柳胜编著. —北京：电子工业出版社，2011.3
（测试实践丛书）
ISBN 978-7-121-12732-8

Ⅰ. ①性… Ⅱ. ①柳… Ⅲ. ①性能试验－软件工具，LoadRunner Ⅳ. ①TP311.56

中国版本图书馆 CIP 数据核字（2010）第 264253 号

责任编辑：李 冰
印 刷：北京中新伟业印刷有限公司
装 订：
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×980 1/16 印张：23 字数：351 千字
印 次：2011 年 3 月第 1 次印刷
印 数：4000 册 定价：50.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。
服务热线：（010）88258888。

什么是软件性能测试?

为什么要做软件性能测试?

如何做软件性能测试?

如果你碰到这三个问题，或者对它们还不太清楚，那么这本书对您来说是一个很好的机会去获得这些问题的答案，本书作者柳胜将和您一起畅游软件性能测试的思海。

《51Testing 软件测试网作品系列》编辑委员会名单

编　委：

王　威（具备多年软件开发经验和软件测试工作经验，51Testing 软件测试培训高级讲师）

王　琰（具有丰富的通信终端产品的测试以及管理工作经验，51Testing 软件测试培训高级讲师）

朴春龙（Mercury 认证 CPC，自动化测试专家，51Testing 软件测试培训高级讲师）

宋　锋（多年软件开发和软件测试工作经验，具备丰富的项目实战经验，51Testing 软件测试培训高级讲师）

陈　霁（ISO 内审员，积累了丰富的测试和管理工作经验，51Testing 软件测试培训高级讲师）

陈文广（谙熟软件测试流程，擅长自动化测试和性能测试，51Testing 软件测试培训高级讲师）

周　峰（信息产业部认证系统分析员，51Testing 软件测试培训高级讲师）

周春江（具备多年通信协议和通信终端设备的测试工作经验，51Testing 软件测试培训高级讲师）

徐林林（熟悉大型应用软件的开发和测试流程；熟悉性能测试流程、方法和工具（如 LoadRunner 等），51Testing 软件测试培训高级讲师）

商　莉（多年从事软件开发、软件测试及质量保证方面的管理工作，51Testing 软件测试培训高级讲师）

邓　强（具有丰富的软件系统测试工作经验，对功能性测试、自动化测试、性能测试及测试框架均有深入的研究，51Testing 软件测试培训高级讲师）

宋光照（具有深厚的开源软件测试实践经验，擅长嵌入式测试，51Testing 软件测试培训高级讲师）

袁艳丽（高级人力资源管理师，具有丰富的计算机培训经验，曾编写教材《系统建模与仿真》，并参与了大连实德等多家企业的大型信息化软件实施和测试工作）

编辑部成员：张晓晓　张燕青

前　言

以工具为切入点，剖析性能测试相关理念；以 LoadRunner 为基础，逐步进阶性能测试实战技能。这是本书上一版到现在始终贯彻的写作风格和宗旨。事实上，广大读者已经通过市场证明我们当时的坚持是正确的，首版一售而光，加印，又售罄，这也是对我们努力工作最大的肯定和鼓励。

从上一版到现在三年的时间里，图书在帮助越来越多的读者进入性能测试领域的同时，包括读者在内的业界同行们也在创造着中国软件性能测试的历史。三年前，性能测试以验证软件系统是否达到相关性能指标为主要目的，而现在，性能测试不仅要找出瓶颈，还要诊断它为什么会存在，甚至进一步预测系统上线后可能在何种情景下会出现性能拐点。这些变化标志着性能测试从定性分析进入到定量分析的时代。同时，定量分析也对我们性能测试人员提出了更高的要求，度量的定义，度量的采集到度量数据的分析，无不深入到被测系统的各个节点的每个层次。这些都需要人员扎实的基本功和广博的知识体系。可以说，LoadRunner 工具层面已经不能满足我们一个高质量、高效率性能测试方案的需求。

面对这些变化和需求，基于上一版内容，本书新增两个较大的篇幅。第 11 章《定量分析及诊断——建立性能度量模型》和第 12 章《测试也开发——自开发性能测试框架》，如果说第 11 章是当前业界高质量性能测试方法策略的总结和归纳，那么第 12 章就是站在业界前沿对性能测试远景的一个前瞻和探索。这两章对于有志于在性能测试领域深造的朋友更具针对性的指导。

同时，在本书出版的同时作者也将和读者建立更多方式的交流渠道。读者可以通过访问作者的个人博客论坛，继续在线下保持长久的联系，获得更多的资源。包括作者定期发布的脚本程序、实战案例、免费框架、相关文档等。

本书的编写背景

LoadRunner 是一个内涵丰富、功能强大的性能测试工具，这是许多专业人士

选择它的原因。初学者面对许多的性能测试概念和 LoadRunner 专业术语，常常感到无所适从，这是因为 LoadRunner 作为软件性能测试的实现者，涉及了软件测试及软件体系架构众多方面的知识。可以说，学习 LoadRunner 是理解和学习软件性能测试的非常好的切入点。

本书既有深厚的性能测试理论的系统完整性，又有非常现实的实践指导意义，还可作为刚进入性能测试领域的测试工程师的入门参考书籍；对已经有一定工作经验的朋友来说，也是一本整理思路、融会贯通的总结笔记。

本书的特色：让复杂变成简单，让简单成为行动

本书的风格为思路清晰，通俗易懂，形象生动，其目的是能让读者迅速理解如下概念。

1．什么是软件性能测试

一场庞大的性能测试演出正在进行，这里没有瞩目的男女主角，都是成千上万涌动的群众演员（并发用户）；没有华丽绚烂的剧本，只有细致缜密的情节安排（性能测试场景）；演至跌宕回肠之处无人喝彩，落幕又意味着新一场演出的开幕（分析调优）。这不是一出梦幻浪漫的咏叹调，更像是一场对百姓平凡生活的写实剧。

只是，这场写实剧还缺少一个最重要的灵魂角色——总导演，那么，你准备好了吗？

2．什么是 LoadRunner

性能测试的庞大演出要能够顺利完成，需要众多的资源，有经验丰富的演员（图灵考验），能够处乱不惊（食客问题），要有逼真的道具（IP 虚拟欺骗），要有宽大的舞台（负载生成器），最重要的是还要保持团队精神和良好的组织纪律（场景创建和监控），并且善于总结和改进（anaylysis 分析器）。

以上括号内的解决方案均由 LoadRunner 厂商提供友情赞助。

3. 本书的思路地图

性能测试中需要明确的问题	本书的答案
软件的性能测试对于你意味着什么	用户眼里的性能（第 1 章） 开发人员眼里的性能（第 1 章） 测试人员眼里的性能（第 1 章） 性能测试策略揭秘（第 1 章）
怎样迅速有效地规划和开展一个性能测试	性能测试最佳实践（GAMEA 模型）（第 1 章） 善其事利其器——工具的选择（第 1 章）
为什么是 LoadRunner	性能测试到 LoadRunner 的映射（第 2 章） LoadRunner 三大高手的联袂演出场景（第 3 章）
怎样使用 LoadRunner 实现你的性能测试目标	脚本语言基础（第 4 章） 玩转 LoadRunner 三大高手（第 5 章 VU、第 6 章 Controller、第 7 章 Analysis）
有没有 LoadRunner 案例可以参考	基于 SMTP 协议的 VU 脚本开发实战（第 8 章） 大型系统性能测试 GAMEA 模型实战（第 9 章） LoadRunner 高级应用技巧（第 10 章）
怎样规划高质量的性能测试	性能测试度量模型（第 11 章）
Loadrunner 已不能满足要求，怎么办	自开发性能测试框架（第 12 章）

4. 你能从本书得到什么

本书旨在培养新一代软件性能测试导演，使你具备如下良好的导演素质：

- 具有领导的能力，表现为能够自如掌控 LoadRunner。
- 具有全局的视角，表现为能够使用 GAME 模型迅速规划并开展一场演出，并且使其顺利完成，达到最好的效果。
- 具有实干的精神，表现为理论与实践相结合，本书有很多的实际案例供你借鉴参考。

关于本书作者

柳胜，硕士研究生专业即为软件性能测试自动化方向，有 8 年的测试工作经验，愿意通过 51Testing 平台将一些经验和体会与测试同行分享，助你成功即是我成功。

目　录

CONTENTS

性能测试从零开始

第 1 章　软件性能测试 ……1

1.1　什么是软件的性能 ……1

1.1.1　软件 ……1

1.1.2　软件性能的产生 ……4

1.1.3　功能与性能的关系 ……4

1.1.4　用户眼里的软件性能 ……6

1.1.5　软件人员眼里的软件性能 ……12

1.2　软件性能测试 ……16

1.2.1　性能测试在软件测试的周期位置 ……16

1.2.2　性能测试策略揭秘 ……20

1.3　如何做性能测试 ……23

1.3.1　Goal（定义目标） ……26

1.3.2　Analysis（分析） ……28

1.3.3　Metrics（度量） ……29

1.3.4　Execution（执行） ……30

1.3.5　Adjust（调整） ……31

1.4　性能测试工具的评估和选择 ……32

1.4.1　测试预算 VS 工具价格 ……33

1.4.2　协议、开发技术、平台、中间件 VS 工具的支持 ……33

1.4.3　工具可使用的复杂程度 VS 项目计划的影响 ……34

第 2 章　第一次亲密接触 LoadRunner ……35

2.1　从性能测试到 LoadRunner 的映射 ……35

2.2　LoadRunner 工作原理 ……38

目　录

CONTENTS

LoadRunner 入门与提升

2.3　安装 LoadRunner ····· 41
2.4　揭开 License 的神秘面纱 ····· 42

第 3 章　走近 LoadRunner ····· 44

3.1　LoadRunner 的运行原理 ····· 45
3.1.1　LoadRunner 三大高手 ····· 45
3.1.2　三大高手联手的一场性能测试盛大演出 ····· 48
3.2　LoadRunner 的录制原理 ····· 50
3.2.1　网络协议与 LoadRunner 的 Vuser ····· 50
3.2.2　选择 LoadRunner Protocol 的两大定律 ····· 55
3.2.3　LoadRunner 录制技术 ····· 56

第 4 章　LoadRunner 脚本语言基础 ····· 62

4.1　C 语言与 LoadRunner 脚本 ····· 66
4.1.1　看不见的 main ····· 67
4.1.2　全局变量与局部变量 ····· 71
4.1.3　在 LoadRunner 脚本里灵活使用 C 语言 ····· 74
4.1.4　高级——用户自定义函数 ····· 78
4.2　通用 VU 函数 ····· 81
4.2.1　事务和事务控制函数 ····· 82
4.2.2　命令行分析函数 ····· 83
4.2.3　系统信息函数 ····· 83
4.2.4　字符串函数 ····· 83
4.2.5　消息函数 ····· 84
4.2.6　运行时（run-time）函数 ····· 84

目　录

CONTENTS

性能测试从零开始

4.3　协议相关函数 …… 85
4.3.1　HTTP 协议原理 …… 85
4.3.2　HTTP 在 LoadRunner 的实现 …… 86

第 5 章　VU——用户行为的模拟器 …… 89

5.1　图灵试验与 LoadRunner VU 模拟奥秘 …… 90
5.1.1　图灵试验场景 …… 90
5.1.2　LoadRunner 模拟揭秘 …… 91
5.2　录制脚本 …… 94
5.2.1　选择协议 …… 94
5.2.2　规划脚本结构 …… 96
5.2.3　HTTP Vuser 中的 URL mode 和 HTML mode …… 96
5.2.4　查看日志 …… 99
5.3　回放脚本 …… 101
5.4　关联 …… 102
5.4.1　录制前 Correlation（关联） …… 103
5.4.2　录制后关联 …… 106
5.4.3　手动关联 …… 109
5.5　脚本视图和树视图 …… 113
5.5.1　树视图（Tree View） …… 113
5.5.2　脚本视图（Script View） …… 114
5.5.3　理解 Snapshot …… 115
5.6　事务、同步点和思考时间 …… 115
5.6.1　Transaction（事务） …… 115
5.6.2　Rendezvous Point（同步点） …… 118

目　录

CONTENTS

LoadRunner 入门与提升

5.6.3　注释 …… 119
5.6.4　Think Time（思考时间） …… 119
5.7　数据驱动——参数化（Parameters） …… 121
5.7.1　为什么需要参数化 …… 121
5.7.2　参数的创建 …… 123
5.7.3　定义参数的属性 …… 130
5.7.4　高级——从已存在的数据库中导入参数数据 …… 136
5.8　检查点（Check point） …… 142
5.8.1　为什么需要检查点 …… 142
5.8.2　检查点实施之一：ContentCheck 定义 …… 144
5.8.3　检查点实施之二：检查函数 …… 144
5.8.4　检查点设置技巧 …… 147
5.9　高级——多 Action …… 147
5.10　高级——脚本错误处理机制 …… 150
5.11　高级——脚本调试技巧 …… 152
5.11.1　动态运行（Animated run）与非动态运行（Non-Animated run） …… 152
5.11.2　日志设置 …… 153
5.12　高级——编写脚本的最佳实践 …… 156
5.13　VU 脚本运行成功后 …… 157

第 6 章　Controller——性能测试的指挥中心 …… 158

6.1　得到细化后的性能需求 …… 159
6.2　设计性能测试场景 …… 162
6.2.1　创建手工场景 …… 164

目　录

CONTENTS

性能测试从零开始

6.2.2　百分比模式创建手工场景 ……173
6.2.3　创建面向目标场景 ……174
6.2.4　多 IP 的实现原理以及模拟 ……180
6.3　运行场景 ……183
6.3.1　场景控制 ……183
6.3.2　执行期间查看场景 ……191
6.3.3　监视场景 ……196
6.3.4　高级——用户自定义数据采集点 ……201
6.3.5　主要的计数器 ……203
6.3.6　实例——分析实时监视图表 ……211
6.4　场景运行后 ……213
6.5　高级——如何合理规划性能测试拓扑 ……213
6.5.1　选择最优的测试机配置 ……215
6.5.2　客户端 CPU 瓶颈估算 ……216
6.5.3　客户端网络瓶颈估算 ……217
6.5.4　客户端内存瓶颈估算 ……219
6.5.5　总评估 ……221

第 7 章　Analysis——寻找系统瓶颈的得力助手 ……222

7.1　Analysis 报告概要（Summary） ……223
7.1.1　概要部分 ……223
7.1.2　统计部分 ……224
7.1.3　事务统计 ……225
7.1.4　HTTP 响应统计 ……227

目　录

CONTENTS

7.2　Analysis 标准图 …… 227
7.2.1　Vuser 图 …… 227
7.2.2　事务图 …… 228
7.2.3　网页细分图 …… 230
7.3　交叉结果和合并图 …… 233
7.3.1　叠加 …… 235
7.3.2　平铺 …… 236
7.3.3　关联 …… 236
7.4　生成测试报告 …… 237
7.4.1　创建 HTML 格式报告 …… 237
7.4.2　创建 Word 格式报告 …… 238
7.4.3　高级——数据点报告 …… 239

第 8 章　欲善其事先利其器——VU 脚本开发实战 … 240
8.1　邮件服务 SMTP/IMAP 协议介绍 …… 241
8.1.1　电子邮件的工作原理 …… 241
8.1.2　SMTP 协议介绍 …… 241
8.1.3　POP3 协议介绍 …… 243
8.1.4　IMAP 协议介绍 …… 244
8.2　VU 的 SMTP Vuser 对 SMTP 协议的封装及实现 …… 244
8.2.1　使用 SMTP Vuser 录制 Outlook 2003 发送邮件 …… 247
8.2.2　对 SMTP Vuser 录制生成的脚本进行分析 …… 251
8.2.3　回放 SMTP Vuser 脚本并分析网络日志 …… 253
8.3　VU Java Vuser 自开发 SMTP 程序 …… 255

目　录

CONTENTS

性能测试从零开始

8.3.1　VU 创建 Java 模板虚拟用户 ····· 255
8.3.2　设置 Java 环境 ····· 257
8.3.3　在 Java Vuser 中开发 SMTP 发送 mail 脚本 ····· 258
8.3.4　参数化增强脚本 ····· 260

第 9 章　学以致用——一步一步做 Web 系统性能测试 ····· 264

9.1　软件系统背景及架构设计介绍 ····· 264
9.2　性能要求和性能指标分析（Goal 阶段） ····· 267
9.2.1　性能指标 ····· 267
9.2.2　业务模型分析和需求细化 ····· 268
9.3　性能测试方案和用例设计（Analysis 阶段） ····· 268
9.4　性能测试各种度量的建立（Metrics 阶段） ····· 273
9.4.1　性能脚本的生成 ····· 273
9.4.2　定义用户行为 ····· 274
9.4.3　场景的设置与运行 ····· 274
9.4.4　计数器的设置与性能数据收集 ····· 276
9.5　运行场景，得到测试结果和相关数据（Execution 阶段） ····· 277
9.5.1　脚本的开发 ····· 277
9.5.2　数据的生成 ····· 278
9.5.3　并发登录测试 ····· 279
9.5.4　负载测试 ····· 279
9.6　分析测试瓶颈（Adjust 阶段） ····· 280
9.6.1　并发测试结果分析 ····· 280

目　录

CONTENTS

LoadRunner 入门与提升

9.6.2　交易流程测试结果分析及性能评价……281
9.6.3　查询流程测试结果分析……281

第 10 章　循序渐进——进阶 LoadRunner 高手……282

10.1　性能测试用例的设计策略……282
10.1.1　“普遍撒网，重点查看”的原则……282
10.1.2　保证数据的有效性……284
10.2　LoadRunner 高级功能的使用——Web Click Vuser……286
10.2.1　Web Click Vuser 的产生背景……286
10.2.2　Web Click Vuser 与传统 Vuser 的差别……286
10.2.3　使用 Web Click Vuser……286

第 11 章　定量分析及诊断——建立性能度量模型…291

11.1　实现性能度量的准备工作……292
11.1.1　性能度量……292
11.1.2　度量方式……294
11.2　案例实践——性能测试第一阶段 Ajax 页面基准性能分析……298
11.2.1　页面基准分析目标……298
11.2.2　分析所使用的工具……299
11.2.3　术语揭示……299
11.2.4　基准测试案例设计及运行……300
11.2.5　结果分析……305
11.2.6　优化建议……306
11.2.7　预测……306

目　录

CONTENTS

性能测试从零开始

11.3　案例实践：性能测试第二阶段 Ajax 系统负载测试 …307
11.3.1　本轮测试目的 …307
11.3.2　测试场景规划 …307
11.3.3　负载测试案例分析 …309
11.3.4　性能结果量化分析 …312
11.4　总结 …315

第 12 章　测试也开发——自开发性能测试框架 …316
12.1　性能测试框架架构设计 …317
12.1.1　技术考虑的准备 …317
12.1.2　Automation Center 性能测试框架设计实例 …319
12.1.3　定义 XML 格式的性能测试任务文件 …325
12.1.4　Automation Center 性能测试调度运行 …329
12.1.5　性能测试报告诊断系统 …330
12.2　核心技术及具体实现方法 …332
12.2.1　性能测试数据源格式定义 …332
12.2.2　测试程序与性能测试框架的接口实现 …336
12.2.3　测试任务克隆技术 …339
12.2.4　多任务通信技术 …340
12.2.5　Observer 性能监控技术 …341
12.2.6　基于 XSL 技术的性能测试报告构建 …344
12.3　总结 …346

附录 A　有关 LoadRunner 的常见问题解答 …347



第1章

软件性能测试

当今，计算机和软件工程发展越来越快，新的概念名词和技术手段层出不穷，可谓日新月异。在软件性能测试范畴内就有很多，诸如并发测试、压力测试、基准测试、测试场景等概念和名词，这让刚接触性能测试的新手眼花缭乱，目不暇接。但我们如果能深入软件性能测试的本质，从哲学的角度看问题，找出其内在联系，比如因果关系、形式内容关系，甚至重叠关系等，理清思路之后，那么做软件性能测试就会如庖丁解牛，游刃有余。

1.1 什么是软件的性能

1.1.1 软件

计算机软件作为人类逻辑智慧的结晶，它可以模拟并替代人类的一些活动，替人“发号施令”。在计算机软件发展的短短几十年内，计算机软件以非常快的

速度渗透到了人类社会的各个角落，比如现在我们在家上网，出门坐公交车刷卡，在工作中发电子邮件等，这些生活的背后都有大量的软件系统运行支持。

同时，有关软件的概念和名词也呈爆炸性增长，从 google 中搜索“软件”关键词，就有 135 000 000 条记录；软件的方向和领域也在不断细化，比如软件架构和平台、软件工程、软件应用，还有软件开发测试等，因此我们可以判断软件的发展趋势是系统化、复杂化，这个趋势使软件能够提供越来越强大的功能，但同时也为我们理解和把握软件带来困难。

但我们做事的原则应该是要把复杂的事情变简单，而不是更复杂，更难理解。当我们试图理解和分析一个复杂的事物的时候，最常用的方法是分而治之，就是要用一个或多个简单的概念去解释或描述这个复杂的事物，这符合我们人类的认知规律，人们对简单的概念能够理解，那么对简单进行综合和归纳，就形成了对复杂的认知。比如，我们想要让一个没有上过网的人明白什么是“电子邮件”，那就可以告诉他“通过网络发送的邮件”，当然他很有可能对“网络”也一头雾水，那么你可以继续向他解释“打鱼的网”。但在软件领域中，我们却经常搞不明白这个道理，一个刚入门有志于软件性能测试的菜鸟小心翼翼地向一个前辈高人请教什么是性能测试，前辈首先以威严的口气告诉他“性能测试是很复杂的”，然后徐徐道来“性能测试分为负载测试、压力测试、容量测试等”。到这里，我相信可怜的菜同学对性能测试已经更加糊涂了，他在请教问题之前，恐怕还能知道性能测试是测试软件性能的，在得到高手回答之后，他开始勤奋地请教 google，没想到 google 回答他的是更多的名词概念（网上文章都是前辈仙人写的）。随着时间推移，菜同学升级成牛同学，他对性能测试名词概念烂熟于心（google 功底深厚），并且牛同学又牢牢坚持与时俱进的思想，独立创新了 *N* 个性能测试概念，并开始向一群小菜粉丝们讲经布道，于是上一幕的画面和对白又开始回放，只是演员变了……这只是一个假想的故事，故事的结局就是通过“学习”，性能测试不仅没有简单，反而越来越复杂了。我们要真正掌握性能测试，那就要避免这样的事情发生，从本质上认识软件性能和软件性能测试。

辩证唯物主义哲学认为，时间和空间是运动着的世间万物的存在形式。大到社会形态，小到个人的活动，都是在一定的空间和时间内进行的。

因此，我们在试图把一件事情表述清楚时，通常要抓住事情的几个关键要素：时间、空间（地点）、人物（主体）、事件。比如“旅行者的一次长途旅行在两个月内从北京到西藏”，这句话中包含了关键要素，其时间是两个月，空间是北京到西藏，人物是旅行者，发生的事件是旅行者在两个月时间范围内发生空间中的转移；又如“一场足球赛”，这个名词看起来简单，但仍清楚地隐含了三个要素，即：时间，通常是 90 分钟（如果没有加时赛和伤停补时）；空间，足球场内；人物，足球运动员，事件就是在足球规则下可能发生的事情，如进球等。

计算机的出现是人类历史上一次伟大的革命，在哲学“物质”这个名词的外延中又多了一个新型事物——计算机软件。如果我们认识到计算机软件也是万物之一，分析其作为“物质”的性质也逃脱不了自然法则的“紧箍咒”，那么我们同样可以把软件作如下简单的理解：

主体：程序，是人类逻辑思维的物化，表现形式为一系列指令代码。

时间：即使计算机速度再快，任何软件程序每一段代码的运行都是需要时间的，例如从用户的感受来讲，就是程序将运行结果响应给用户的速度。

空间：软件运行的环境，以资源的方式存在，通常是软件以间接或直接的方式占用并使用硬件资源和其他软件资源。

硬件资源主要指运行该软件的硬件平台，有 CPU、内存和存储系统等，如果软件是基于网络架构的，那么硬件还有网络硬件，如交换机、路由器等。

软件资源包括操作系统、开发平台、中间件和数据库等，它们以库文件和 API 的方式提供给应用软件使用。

事件：软件按照用户的要求运行，运行的同时必然要占用时间资源和空间资源。

由于软件代码是人的逻辑思想的表现，所以软件在设计思想和实现方法上也有很大差异。另外，随着软件的发展，产生了各种应用领域的软件，它们之间存在着千丝万缕的关系。从层次上看，有系统软件，应用软件和介于两者之间的中

间件。因此一个软件的运行牵涉的因素很多，需要从各个方面分析。

1.1.2 软件性能的产生

用户能够看到的是软件越来越通用，功能越来越庞大，从哲学角度上看待软件本身，其发展是一个从简单到复杂，从低级到高级，从无序到有序的过程。

在计算机发展的初期，计算机软件对硬件有很强的依赖性，而且还没有广泛的通用性，只有少数的个人或机构才使用软件这个“奢侈品”，当时用户也没有软件性能的概念，通常为了实现软件的功能而不计一切代价。比如，1946 年 2 月 15 日，世界上第一台通用电子数字计算机“埃尼阿克”（ENIAC）在美国研制成功。它当时由 1.8 万个电子管组成，是一台又大又笨重的机器，体重达 30 多吨，占地有两三间教室般大。它当时的运算速度仅为每秒 5000 次加法运算，在现在看来，它占用如此多的资源，又运行得如此慢，在当时却是相当了不起的成就，因为它已经实现了功能——能够做加法运算。可见初期的软件是简单的，当时用户的要求用现在的眼光来看真有点可怜巴巴，对软件的要求不高，只要能工作就 OK 了。

软件诞生后，短短几十年，软件业奇迹般的高速发展，逐渐走下了高高在上的神坛，广泛应用到人类社会的各个领域，用户也不再把软件看做神秘的玩意，而是普通的商品，**开始从经济学的角度来考虑软件产品，这是一个意味深长的变化**。讲经济就是要运用投入产出的关系分析和指导软件工程的各种活动和环节，软件运行不能以硬件不计成本为假设，要尽可能地少占用各种硬件资源，同时，软件运行的速度也要尽可能地快，每秒 5000 次加法运算是根本不可想象的，也是不可能被用户接受的。**这些其实就是用户的最原始的性能需求**。

1.1.3 功能与性能的关系

首先，软件的性能和功能的源头都是来自于用户的需求。

功能指的是在一般条件下软件系统能够为用户做什么，能够满足用户什么样的需求。拿一个电子邮件系统来讲，用户期望这个软件系统能够提供收发电子邮

件、保存草稿、设置偏好等功能，只有这些功能实现了，用户才认为这是他想要的软件。但是随着软件市场竞争的激烈，软件技术的日益提高，系统能不能工作已经是一个最起码的门槛，能够“又好又快”才会得到用户的青睐，而性能则是衡量软件系统“好快”的一个重要考虑因素。“好”就是要为用户省钱，用最小的硬件成本运行软件系统；“快”就是软件响应时间要短，我们的用户都是急性子，最好一秒钟也不要等。简单地说，性能就是在空间和时间资源有限的条件下，软件系统还能不能工作。

如果把上面邮件的功能和性能需求量化，写成用户需求说明书可能是下面这个样子：

功能：

邮件系统能够支持收发以 30 种语言为标题和正文的邮件，并支持粘贴 10MB 的邮件附件。

性能：

邮件系统能够在 2GB RAM/1GHz CPU 的服务器上，支持 10000 注册用户，日均处理 10000 邮件，响应时间不超过 5 秒/封。

我们来对比一下功能需求说明和性能需求说明，发现两者有一些不同之处：

（1）功能需求中名词和动词多，描述软件主体和动作行为，比如“标题”、“正文”、“收发”、“粘贴”等；

（2）性能需求中对涉及容量和时间词汇多，如“2GB RAM 服务器”、“1000 注册用户”、“5 秒/封”等。

相信我们的读者已经从上面的对比看出功能和性能的区别了，软件性能和功能区别的实质是，软件功能焦点在于软件“做什么”，关注软件物质“主体”发生的“事件”；而软件性能则关注于软件物质“做得如何”，这是综合“空间”和“时间”考虑的方案（资源和速度），表现为软件对“空间”和“时间”的敏感度。认识到性能的这个基本特征对于性能测试人员非常重要，因为在下面的章

节中我们将要通过多个“空间”和“时间”的组合，来揭开性能指标的实质和提高的办法。另外，我们也要认清一个事实，软件的性能实现是建立在功能实现的基础之上的。

这就像一个人首先要能跑，这是一个健康的人的正常功能，然后才能参加百米比赛，这就如对人身体的性能考验。而百米比赛隐含了两个要素：一个是运动员有一个一百米的运动空间；另一个是比赛，要跑得足够快，要在短时间内跑完。因此我们说百米比赛其实就是一个空间和时间的综合结果。

“空间”和“时间”是一个哲学中抽象层次较高的概念，在不同的应用范围有不同的诠释。那么在软件理论和实践中，我们怎样理解“空间”和“时间”呢？所谓“仁者见仁，智者见智”，下面我们就分别从用户的角度和软件人员的角度来看一下软件的性能。

1.1.4 用户眼里的软件性能

软件系统在满足用户强大的功能需求同时，架构和实现上也变得复杂，软件系统经过单机系统时代、客户机服务器系统时代，到现在跨广域网的庞大分布式系统时代，这样的例子在金融、电信系统中随处可见。

系统的业务量大了，就要使用更多的时间和空间资源，在一般情况下不能出现的软件性能问题就暴露出来了，这些问题“不鸣则已，一鸣惊人”，轻则让软件对外不能正常提供服务，重则可能会导致系统的崩溃甚至数据的丢失，这都会给用户带来无法估量的损失。



案例分析

某西部大型油田使用钻井平台数据采集系统，在上线之前已经通过功能测试，但软件系统上线之后，在使用采集的电子数据勘探油层时，总是不能准确地找到油口，导致数百万元的损失。经过研究试验，发现软件从平台采集的数据和手工采集的数据有很大出入，性能测试后，找到根本原因：由于采集过程中产生的数据量非常大，导致软件系统在采集过程中线程死掉，丢失部分数据，最终产生的是一个错误的采集结果，为工程人员提供了错误的判断依据。

案例分析

日本第三大手机运营商——软银移动 2006 年 10 月遇到了麻烦，本指望通过降低手机资费来吸引用户，谁想大量用户蜂拥而至却导致自己的电脑系统陷入瘫痪，软银移动在 10 月 29 日不得不宣布暂停接纳新的用户，直接损失逾亿日元。

用户当然不想看到以上的场景发生在自己的软件系统上，“瘫痪”意味着响应时间过长，不能为客户正常提供服务；数据丢失则是一个不可接受的严重问题，损失几乎不可弥补。因此用户对软件性能的要求日益细化严格，可以说是“与时俱进”。

简单地说，在软件发展的初级阶段，“又要马儿跑，又要马儿少吃草”，这是当时很多用户对软件系统提出的性能要求，“跑”有关时间，“草”有关空间。马儿跑，就是软件系统给用户的响应要快，处理时间要短；马儿少吃草就是软件系统能够尽可能地少占用和消耗资源，诸如内存、CPU 等。因此，测试人员在做性能测试时，往往要把响应时间、内存利用率、I/O 占用率等写在最后测试报告里，因为这是用户最关心的东西。

随着用户的软件质量意识的增强，用户对软件的性能需求也越来越多，越来越细致。这时不仅要让马儿跑，还要马儿能快能慢（软件系统的伸缩性），“路遥知马力”（软件系统在长时间运行下的稳定性）等。细数起来，如下：

- 计算性能；
- 资源的利用和回收；
- 启动时间；
- 伸缩性；
- 稳定性。

计算性能——就是马儿要能跑，要有很快的速度，最好是“日行千里，夜行八百”。对软件系统来讲，计算性能是用户最关心的一个指标，即软件系统有多快。比

如，用户会关注软件系统执行一个典型的业务需要花多少时间。我们要给出用户答案，我们的系统完成用户典型操作，比如业务的交易计算，数据的增、删、改、查时间是不是在用户可以接受的范围内。

资源的利用和回收——就是马儿少吃草。软件系统的“草料”就是其依存的硬件和软件资源，硬件资源包括客户端硬件、服务器硬件和网络硬件；软件资源包括操作系统、中间件和数据库等。

其中要特别说的是，运行软件系统需要使用到的服务器内存数量，对于整个系统的性能表现是至关重要的。因此，软件系统能否在运行时有效地使用和释放内存是我们考察软件性能的一个重要因素。

对计算机来讲，计算机内存为程序提供运行空间（有代码区和数据区），如果内存不够大，CPU 就不能把全部的数据和程序放到内存里，只好放一部分在内存，一部分放在硬盘中，现用现取，而读取内存和读取硬盘数据的速度要差好几个数量级，这就大大影响了计算机的工作效率。如果还不能理解内存的重要性的话，可以用个形象的例子来说明：

如果 CPU 是个画家，那么内存就是他的工作台。工作台上放着画布（被操作的数据），还有各种画笔、刷子等各种工具（运行的程序）。如果工作台（内存）不能足够大，容纳不下绘画所使用的所有工具，那么画家就需要不时地去储藏室（硬盘等存储设备）里取所需的工具，这就会大大影响绘画的速度。

所以在评价一个系统性能的时候，要特别关注这个系统对内存的使用。

启动时间——这是马儿的加速度问题。用户希望系统进入正常工作状态的时间越短越好，尤其在主备系统中，软件的启动时间直接影响主备的切换效率。而不同软件系统启动时间会不同的。J2EE 系统在第一次启动的时候一般会比较慢，因为期间涉及缓存的加载、JSP 页面的编译、Java class 编译成机器指令等。所以在第一次启动应用感到非常慢是比较正常的，这也是 J2EE 或者 Java 应用的一个特点。而 C/C++程序直接运行的是二进制机器代码，启动速度就要快一些。

伸缩性——马儿要能快能慢。伸缩性是分析系统性能经常被忽略的一个方面。比如一个系统在 50 个并发用户访问的时候表现正常，但是当并发用户达到 1000

的时候，系统表现如何？服务器的性能是逐渐下降呢，还是在某个拐点附近急剧下降呢？

如图 1-1 所示，该图是一个伸缩性不好的系统的表现，随着并发用户的增加，平均相应时间越来越长。系统最终会达到一个不可用的程度，没有一个用户会接受系统这样的性能表现。

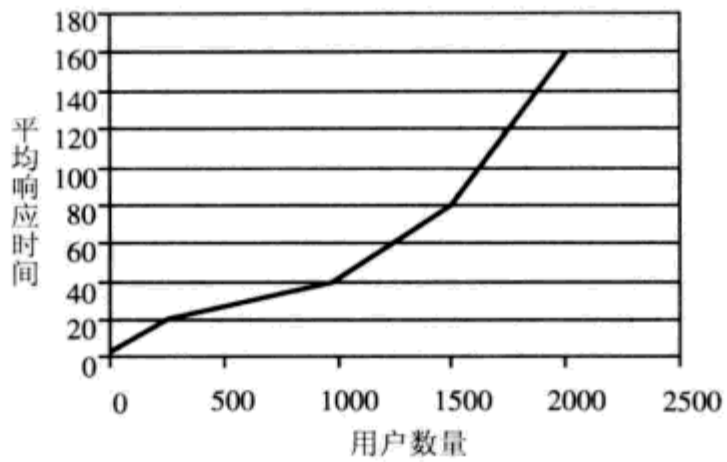


图 1-1　伸缩性不好的用户 - 响应时间图

如图 1-2 所示就是一个伸缩性较好的系统的表现，随着并发用户的增加，平均响应时间逐渐稳定下来。

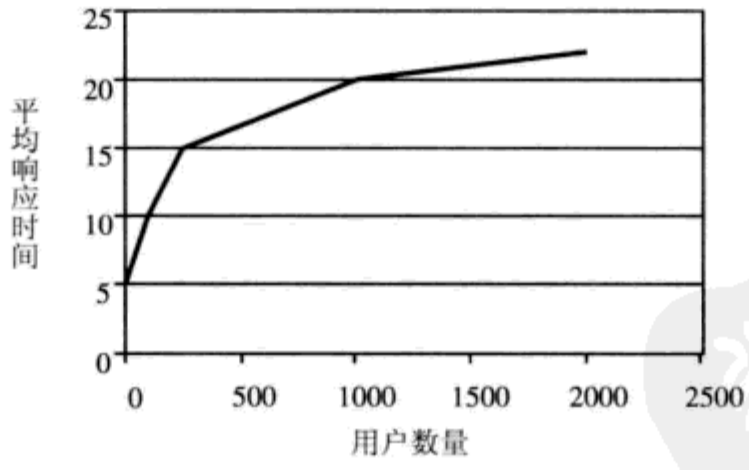


图 1-2　伸缩性良好的用户 - 响应时间图

稳定性——千里马能够“路遥知马力”，而黑马只能够一时跑得快。用户希望自己的软件系统是千里马，而不是黑马。尤其是金融和电信系统，这些系统基本上都是每天 24 小时运转，时时刻刻准备着为用户提供服务。如果它们在运行一段时间后出现了问题，不能响应用户的请求甚至破坏或丢失了数据，那么系统为

用户带来的损失是巨大的。这种稳定性问题应该在软件系统上线之前就被考虑并得到解决。

“快”、“好”这只是用户的主观体验，如果能让这些感觉和要求被其他人正确地理解（尤其是对软件人员），那么就需要用数据把上述用户的感受量化并表达出来，这就是性能指标。

通常，衡量一个软件系统性能的常见指标有：

1. 响应时间（Response time）

响应时间就是用户感受软件系统为其服务所耗费的时间，对于网站系统来说，响应时间就是从点击了一个页面计时开始，到这个页面完全在浏览器里展现计时结束的这一段时间间隔，看起来很简单，但其实在这段响应时间内，软件系统在幕后经过了一系列的处理工作，贯穿了整个系统节点。根据“管辖区域”不同，响应时间可以细分为：

（1）服务器端响应时间，这个时间指的是服务器完成交易请求执行的时间，不包括客户端到服务器端的反应（请求和耗费在网络上的通信时间），这个服务器端响应时间可以度量服务器的处理能力。

（2）网络响应时间，这是网络硬件传输交易请求和交易结果所耗费的时间。

（3）客户端响应时间，这是客户端在构建请求和展现交易结果时所耗费的时间，对于普通的瘦客户端 Web 应用来说，这个时间很短，通常可以忽略不计；但是对于胖客户端 Web 应用来说，比如 Java applet、AJAX，由于客户端内嵌了大量的逻辑处理，耗费的时间有可能很长，从而成为系统的瓶颈，这是要注意的一个地方。

那么客户感受的响应时间其实是等于**客户端响应时间+服务器端响应时间+网络响应时间**。细分的目的是为了方便定位性能瓶颈出现在哪个节点上（何为性能瓶颈，下一节中介绍）。

2. 吞吐量（Throughput）

吞吐量是我们常见的一个软件性能指标，对于软件系统来说，“吞”进去的是请求，“吐”出来的是结果，而吞吐量反映的就是软件系统的“饭量”，也就是系统的处理能力，具体说来，就是指软件系统在每单位时间内能处理多少个事务/请求/单位数据等。但它的定义比较灵活，在不同的场景下有不同的诠释，比如数据库的吞吐量指的是单位时间内，不同 SQL 语句的执行数量；而网络的吞吐量指的是单位时间内在网络上传输的数据流量。吞吐量的大小由负载（如用户的数量）或行为方式来决定。举个例子，下载文件比浏览网页需要更高的网络吞吐量。

3. 资源使用率（Resource utilization）

常见的资源有：CPU 占用率、内存使用率、磁盘 I/O、网络 I/O。

我们将在 Analysis 结果分析一章中详细介绍如何理解和分析这些指标。

4. 点击数（Hits per second）

点击数是衡量 Web Server 处理能力的一个很有用的指标。需要明确的是：点击数不是我们通常理解的用户鼠标点击次数，而是按照客户端向 Web Server 发起了多少次 http 请求计算的，一次鼠标可能触发多个 http 请求，这需要结合具体的 Web 系统实现来计算。

5. 并发用户数（Concurrent users）

并发用户数用来度量服务器并发容量和同步协调能力。在客户端指一批用户同时执行一个操作。并发数反映了软件系统的并发处理能力，和吞吐量不同的是，它大多是占用套接字、句柄等操作系统资源。

另外，度量软件系统的性能指标还有系统恢复时间等，其实凡是用户有关资源和时间的要求都可以被视作性能指标，都可以作为软件系统的度量，而性能测试就是为了验证这些性能指标是否被满足。

1.1.5 软件人员眼里的软件性能

用户恨不能让软件有无限的性能，但作为软件技术人员，我们需清楚地认识到，那种理想化的要求是不可能的。在软件性能方案中，没有什么万能钥匙，软件性能方案充满了辩证的各种矛盾。每种方案和方法几乎都有利有弊。只有把握设计系统的具体环境，明确设计目标，具体问题具体分析，合理平衡各种矛盾，牢牢抓住主要矛盾，才能产生出优化的软件系统性能方案。

在上面的分析中，我们得知软件性能是软件运行空间和时间综合考虑的解决方案。那么其实满足用户的性能需求，只有以下几种方案：

1. 消除软件对空间和时间不必要的浪费

一个最明显的例子就是内存泄漏问题，它被开发人员看做是大忌。

严格地说，内存泄漏应该属于软件程序设计的一种缺陷，该缺陷直接导致了程序在运行过程中无法释放不再需要的内存空间，从而造成内存资源浪费，严重的会造成无可用内存，导致系统崩溃。具体来说，当用户程序在运行过程中需要动态获得内存时，操作系统总是从堆（heap）上分配相应的空间给应用，分配的结果是将该堆内存的起始地址通过指针返回给应用。正常情况下，使用完这块内存后，应通过系统调用主动通知操作系统回收这些堆内存以便重用。但是，如果由于设计缺陷导致在某些情况下程序没有主动地通知到操作系统，而后应用又失去了对这块内存的引用时，则该堆内存块将成为既不受程序控制，又不能被系统回收重用的“孤儿”内存，这便是我们所指的内存泄漏。



案例分析

```
void foo( )
{
   char *str;
   str = (char*)malloc(32*sizeof(char));
   strcpy(str, "hello world");
   return;
  /* str 所指向的 32 个字节的内存没有被释放，当 foo()返回时造成内存泄漏 */
```

```
    }
```

解决：C 语言中 malloc 和 free 函数要配对使用。



案例分析

```
    void foo()
    {
    //定义 string1 指针，其指向一个堆上的 100 个字节的内存空间
    char *string1 = (char*)malloc(100*sizeof(char));
    //定义 string2 指针，其指向一个堆上的 200 个字节的内存空间
    char *string2 = (char*)malloc(200*sizeof(char));
    scanf("%s", string2);
    string1=string2;/*string1 原先指向的 100 个字节的内存没有被释放*/
    /*而后又被指向 string2 所指的内存块，造成前面 100 个字节的内存泄漏*/
    free(string2);
    free(string1); /* 这个 free()调用会失败，因为 string1 指向的内存地址与
string2 的相同，而那块内存已经被释放了 */
    return 0;
    }
```

解决：在程序堆上分配内存后，要在使用完后及时释放，同时避免野指针的产生，比如 string1。

原理：内存是软件运行的重要的空间资源，内存泄漏实际上是浪费了软件的空间资源。因此，内存泄漏对软件的性能影响十分重要。

另外，对于程序在时间上的浪费，我们通常是采用优化算法和数据结构的解决策略。



案例分析

最近几年，很多知名软件公司在招聘软件测试人员，考察代码能力的时候，内存泄露和算法优化是经常的试题之一。这说明了用户对软件性能的要求越来越严格，已经传递到了软件公司。

2. 以空间换时间

软件的高性能并不是凭空产生的，在解决了空间和时间浪费的问题之后，如果用户还有更高的性能要求，我们软件人员只好“偷梁换柱”，做一下调整，而这种调整往往是很灵活的。

空间换时间是软件人员解决性能问题最常见的方法。是在系统功能正常的前提下增大软件空间开销的方法来缩减运行的时间。一般的方法有算法调整、并行计算方法、体系结构方法和一些不是“办法”的办法。

通常的解决方案有 Cache 缓存、数据库的 index 等。



案例分析

一个动态网站服务器总发生 CPU 耗尽的问题，因此造成给用户的响应缓慢或者长时间没有响应，进而引起 Server 的宕机。经调查分析，网站首页是个 PHP 程序，每次用户访问都要多次查询数据库，也没有 Cache 机制，数据库查询负荷过高，耗尽 CPU。

解决：改写网站首页以及部分频繁访问的程序，增加 Cache 机制，减少数据库访问。

原理：将常用数据放在服务器的内存中，虽然增加了内存的开销，但带来了时间上的优化，对用户而言，提高了处理速度。

3. 以时间换空间

时间换空间的方案解决性能问题的情形比较少。有时会出现在对内存要求十分苛刻的地方，比如嵌入式操作系统中。



案例分析

程序设计的要求是不设中间变量，交换两个变量的值。

我们通常的中间变量的解决方案是：

```
Void swapOne(int *a, int *b)
{
   Int temp;
   Temp = *a;
   *a = *b;
 *b= temp;
}
```

但这里需要在程序中为 temp 变量在栈上分配一个空间。可不可以不用这个 temp 变量呢？

解决：

修改程序如下：

```
void swapTwo(int *a,int *b)
{
*a=*a+*b;
*b=*a-*b;
*a=*a-*b;
}
```

原理：修改之后，多了运算复杂度，但没有使用第三方变量，减少了空间的占用。

以上是我们从简单的程序例子来理解性能解决方案，但现实要远远复杂得多，因为随着软件系统功能的复杂强大，软件的规模也在不断扩大，我们不可能完全自己开发程序，很多时候是利用已有的平台和中间件资源。在这种场景下，我们应该怎样考虑性能问题呢？

第一，软件系统设计的架构及技术平台

软件在设计阶段一旦决定采用哪种架构和技术，其性能也就注定只能在一定的范围内变动了。这就是“先天”因素。比如在上节讲到的一个删除/增加数据的业务操作，如果用户对时间非常苛刻，密集型计算、在线的大数据量统计和分析等应用，这些场景通常 J2EE 不能够很好地解决，使用 C++或者其他平台搭建会更合理些。如果在这些场景下硬要采用 J2EE 架构，那么开发和设计人员如何绞尽脑汁，优化设计和程序，也很难满足用户的性能要求。

第二，中间件的设置和优化

这里的中间件是广义的中间件，是应用程序调用的第三方软件，包括操作系统、数据库、Web 服务器、消息服务器等。我们不能改变中间件的程序，只能通过调优手段来提高它所支持的软件系统的性能。

第三，硬件的配置

这里包括服务器硬件配置和网络环境。服务器硬件包括内存、CPU 等，网络环境有交换机、路由器等。

1.2 软件性能测试

在上一节中，我们知道软件系统的性能问题多种多样，这给用户带来巨大的风险，那么我们如何能够在软件系统上线之前，找出软件中潜在的性能问题呢？目前软件性能测试是发现软件性能问题最有效的手段，而完备有效的性能测试是最关键的，在本节中我们将从流程和技术的角度解析如何构建一个高效的性能测试模型。

1.2.1 性能测试在软件测试的周期位置

首先，软件性能测试属于软件测试范畴，存在于软件测试的生命周期中。一个软件的生产过程通常遵循 V 型图，如图 1-3 所示。

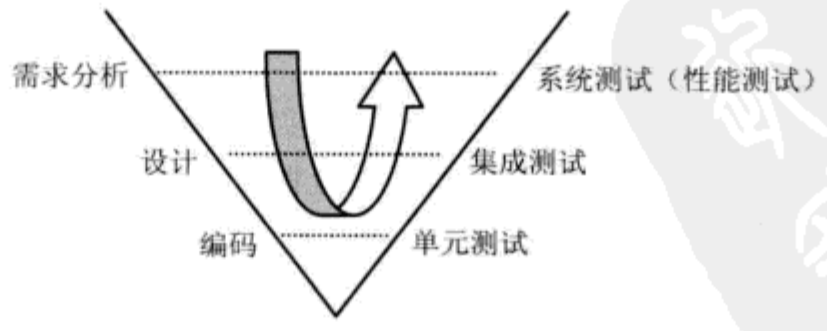


图 1-3 软件开发—测试 V 型图

在通常的软件生产周期中，先由用户提出用户需求或经系统分析核定以后提出系统需求，开发人员再经过需求分析提出软件需求规格说明，进行概要设计，提出概要设计说明，进行详细设计，提出详细设计说明，最后就是对每个模块进行编码。到测试阶段，测试按照开发过程逐阶段进行验证并分步实施，体现了从局部到整体、从低层到高层逐层验证系统的思想。对应软件开发过程，软件测试步骤分为代码审查、单元测试、集成测试、系统测试。

而性能测试就属于软件系统级测试，其最终目的是验证用户的性能需求是否达到，在这个目标下，性能测试还常常用来做：

（1）识别系统瓶颈和产生瓶颈的原因；

（2）最优化和调整平台的配置（包括硬件和软件）来达到最高的性能；

（3）判断一个新的模块是否对整个系统的性能有影响。



系统瓶颈：

瓶颈本来是指玻璃瓶中直径较小并影响流水速度的一段，用它来比喻软件系统中出现性能问题的节点是很形象的，比如一个典型的分布式系统架构如图 1-4 所示。

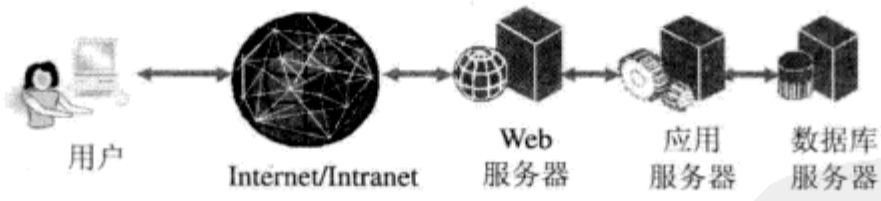


图 1-4　软件系统压力流动图

如果把软件系统看做是交通系统，那么网络就是一条条大道，客户端、防火墙、负载均衡器、Web 服务器、应用服务器（中间件）、数据库等各个系统节点就是交通要塞，客户的请求和数据就像在道路上行驶的车辆，如果在某处发生堵车，那么整个交通系统都会不畅。在这个时候，我们就要分析是哪里出了问题，是道路不够宽，还是某处立交桥设计不合理而引起堵塞等。找到问题

的关键点，那么此关键点就是本系统的瓶颈。软件系统也是如此，我们做性能测试的大部分工作都是为了寻找这个瓶颈到底在何处。

需要注意的是，**软件的性能瓶颈可能不止一处。**

作为软件测试的一种，软件测试的规则同样适用于性能测试中：

（1）确定预期输出是测试必不可少的一部分

如果事先无法肯定预期的测试结果，往往会把看起来似是而非的东西当作正确结果。必须提倡用事先精确对应的输入和输出结果来详细检查所有的输出。对于性能测试来说，预期输出就是用户的性能需求，一份明确的性能需求是成功性能测试的先决条件。

（2）必须彻底检查每一个测试结果

事实上，在最终发现的错误，有相当一部分在前面的测试中已经暴露出来了，然而由于人们未能细心检查先前的测试结果而遗漏了。

一段程序中存在的错误概率与在这段程序中发现的错误数呈正比。

这是 pareto 原则应用于软件测试，也包括性能测试，即性能测试发现的错误中的 80%很可能集中在 20%的程序模块中。

（3）穷举测试是不可能的

在性能测试中不可能覆盖每一个功能部分，这也意味着有性能问题的模块可能被忽略掉，这样的话，我们在设计性能测试案例时，应该采取一些策略和技巧，使用尽可能少的性能测试用例，发现尽可能多的 bug。这方面内容我们将在本书的第 10 章中介绍。

在具有软件测试共性的同时，性能测试也有自身的一些特点。

1．性能测试不是功能测试

性能测试不要求也无法做到覆盖软件所有的功能，通常我们只是对系统中某

些功能或模块做性能测试。一般的，我们在选择性能测试案例时需要遵循以下的原则：

（1）基本且常用的

比如，一个 E-mail 系统，基本且常用的功能有注册、登录、收邮件、查询邮件，用户使用这些功能的频率较高，要做性能测试。而高级查询、过滤器、邮件列表等功能被使用的次数较少，就可以不做性能测试，或者进行性能测试的优先级低一些。

（2）对响应时间要求苛刻的

这样的要求经常出现在金融和电信等对实时性要求比较高的系统中。比如，从手机呼叫开始，经过基站、核心网，再到被叫手机响铃，整个系统的处理时间应该在用户能接受的范围内。另外，一个负责和手机通信的基站在发生故障或掉电后，要能很快地恢复工作状态。这些功能都对时间有着严格的要求，一定要做性能测试，当然实际运作时，电信系统上线时所做的性能测试不仅仅限于这些功能。

将这些功能细分就是性能测试中的事务（Transaction）。关于事务这个概念我们在后面的章节中将详细阐述。

2. 性能测试属于系统级测试

从 V 型图可以看到，性能测试属于系统级测试。那么性能测试是基于单元测试、集成测试、功能测试等都已经完成的基础上，站在用户的角度去测试整个系统的。这包含两个含义：

第一，性能测试是“两头在外”，软件性能需求不仅直接来自用户，最终目的也是服务于用户。从上面我们讲到用户的需求和性能测试指标的对应关系，就可以看出这个关系。所以在性能测试开始之前，一般要搭建模拟生产环境，分析用户规模及使用场景等。

第二，性能测试开始的必要条件是软件系统已经处于一个比较稳定的状态，系统架构、主要代码、中间件等都不再有大的变化，否则会给性能测试带来很大

的风险。

? 思考

基于以上事实，我们应该在软件流程什么阶段开始性能测试？结合自己的实际工作进行分析。

1.2.2 性能测试策略揭秘

谈到“策略”，这是如今很火、使用较多的一个词。不光在 IT 领域，其他各个行业中也都有各种各样的策略，如营销策略、风险规避策略等。

策略即谋略、手段、方法，表现为权宜的行动路线、指导原则或过程。

做事情讲策略，这是一种智慧，是人们聪明起来的表现，但当越来越多的策略“概念化”的时候，我们不得不去思考我们到底要达到什么样的目标，什么样的策略才是我们需要的。

Case 案例分析

引用网上一位哲人说的话：“概念只是为了方便人们理解和研究世界万物事物而制造的工具，而最终结果将使概念不再需要，就如同庄子所说的得意而忘言”。语言就是一种包装材料，它包装的是某种含义。因为人类传递信息必须使用语言，所以我们在研究的时候不得不借助于这种包装，但是当人的思维能力具备了打开包装直接取得内部的含义的时候，语言就变得多余了。这时候再关注于语言和概念本身就成了买椟还珠的现代版了。

因此，我们应该关注的不是概念本身，而是概念背后的含义。理解了含义，再冠予它什么样的名词头衔，如“攻略“，对于我们都无关紧要了。而理解一个概念，我们可以靠 WWH 方法，即对概念的三个问题：Why、What、How。

好，言归正传，回到软件性能测试策略中来。在性能测试过程中，只要有事情做，就会有策略，如设计用例有设计策略，执行时有执行策略，调优时还有调优策略。为了不产生混淆，我们要说明的是，在本节中讨论的策略是性能测试设

计策略。

Why（为什么会有不同的策略）

在软件性能一节中，我们看到软件的性能来自软件对空间和时间的综合方案，这种组合是很多的，因此用户的软件性能需求可能会多种多样。对于软件人员，我们做性能测试也要因地制宜，根据不同的性能需求，选择不同的测试策略。

What（什么是性能测试设计策略）

验证性能需求是测试目的，测试策略即已经被证明是行之有效的测试方法。

How（怎样实施）

常见的性能测试方法有以下几种。

1．负载测试

在这里，负载测试指的是最常见的验证一般性能需求而进行的性能测试，在上面我们提到了用户最常见的性能需求就是“既要马儿跑，又要马儿少吃草”。因此负载测试主要是考察软件系统在既定负载下的性能表现。我们对负载测试可以有如下理解：

（1）负载测试是站在用户的角度去观察在一定条件下软件系统的性能表现。

（2）负载测试的预期结果是用户的性能需求得到满足。此指标一般体现为响应时间、交易容量、并发容量、资源使用率等。

2．压力测试

压力测试是为了考察系统在极端条件下的表现，极端条件可以是超负荷的交易量和并发用户数。注意，这个极端条件并不一定是用户的性能需求，可能要远远高于用户的性能需求。可以这样理解，**压力测试和负载测试不同的是，压力测试的预期结果就是系统出现问题，而我们要考察的是系统处理问题的方式**。比如说，我们期待一个系统在面临压力的情况下能够保持稳定，处理速度可以变慢，

但不能系统崩溃。因此，压力测试是能让我们识别系统的弱点和在极限负载下程序将如何运行。

案例分析

负载测试关心的是用户规则和需求，压力测试关心的是软件系统本身。对于它们的区别，我们可以用华山论剑的例子来更加形象地描述一下。如果把郭靖看做被测试对象，那么压力测试就像是郭靖和已经走火入魔的欧阳峰过招，欧阳锋蛮打乱来，毫无套路，尽可能地去打倒对方。郭靖要能应对住，并且不能丢进小命。而常规性能测试就好比郭靖和黄药师、洪七公三人约定，只要郭靖能分别接两位高手一百招，郭靖就算胜。至于三百招后哪怕郭靖会输掉那也不用管了。他只要能做到接下一百招，就算通过。

思考

我们在做软件压力测试时，往往要增加比负载测试更多的并发用户和交易，这是为什么？

3. 并发测试

验证系统的并发处理能力。一般是和服务器端建立大量的并发连接，通过客户端的响应时间和服务器端的性能监测情况来判断系统是否达到了既定的并发能力指标。负载测试往往就会使用并发来创造负载，之所以把并发测试单独提出来，是因为并发测试往往涉及服务器的并发容量，以及多进程/多线程协调同步可能带来的问题。**这是要特别注意，必须测试的。**

4. 基准测试

当软件系统中增加一个新的模块的时候，需要做基准测试，以判断新模块对整个软件系统的性能影响。按照基准测试的方法，需要打开/关闭新模块至少各做一次测试。关闭模块之前的系统各个性能指标记下来作为基准（Benchmark），然后与打开模块状态下的系统性能指标作比较，以判断模块对系统性能的影响。

5. 稳定性测试

“路遥知马力”，在这里我们要说的是和性能测试有关的稳定性测试，即测试系统在一定负载下运行长时间后是否会发生问题。软件系统的有些问题是不能一下子就暴露出来的，或者说是需要时间积累才能达到能够度量的程度。为什么会需要这样的测试呢？因为有些软件的问题只有在运行一天或一个星期甚至更长的时间才会暴露。这种问题一般是程序占用资源却不能及时释放而引起的。比如，内存泄漏问题就是经过一段时间积累才会慢慢变得显著，在运行初期却很难检测出来；还有客户端和服务器在负载运行一段时间后，建立了大量的连接通路，却不能有效地复用或及时释放。

6. 可恢复测试

测试系统能否快速地从错误状态中恢复到正常状态。比如，在一个配有负载均衡的系统中，主机承受了压力无法正常工作后，备份机是否能够快速地接管负载。可恢复测试通常结合压力测试一起来做。



每种测试有其存在的空间和目的。当我们接手一个软件项目后，在有限的资源条件下，选择去做哪一种测试，这应该根据当前软件过程阶段和项目的本身特点来做选择。比如，在集成测试的时候要做基准测试，在软件产品每个发布点要做性能测试。

1.3　如何做性能测试

一个项目要取得成功是困难的，因为成功的项目需要多个因素和条件来支持；而一个项目失败却很容易，只要若干因素之中的一个出现问题，就有可能导致项目失败。比如中途测试人员发生变化，性能指标未和用户达成统一理解等。笔者还曾看过一个例子，因为测试报告的格式与用户要求的格式不一致，而不得不重新再执行一次所有的性能场景，来采集用户要的数据。

实际上，当我们做过的性能测试项目越多，就会发现越多的因素可能会影响性能测试项目的成败，甚至可以是千奇百怪的。

在本节中，我们主要是寻找出不同性能测试项目的共性，而总结出一个具有普遍意义的性能测试过程。遵循过程做性能测试，在大多数情况下可以有效地规避风险，并能取得比较好的性能测试结果。这当然不是意味着我们有了这个过程，就不考虑其他因素了，只是说每个项目都会有自己的独特因素要考虑，尽管这些因素可能很重要，但它们并不在本节的讨论范围内。

在给出此过程模型之前，我们要澄清两点事实：

第一，性能测试过程从何时开始，又在何时结束？

这是一个基本而重要的问题。

在各种书籍和资料中，有关性能测试过程的描述不尽一样：

比如 LoadRunner 手册中提供的过程是：**计划测试→测试设计→创建 VU 脚本→创建测试场景→运行测试场景→分析结果**。

而在 Segue 中提供的性能测试过程，是一个 try-check 过程，即：**评估需求→开发测试→建立基线→执行测试→分析结果→回归测试→测试结束**。

上面 LoadRunner 和 Segue 描述各自的性能测试过程最大的区别不在于工具部分，而是在于两者过程的入口和出口条件不一致。这使得它们其实在描述两件事情，或者说是在描述一个事情的两个部分。

在 CMM 中，软件测试和软件设计、编码一样，隶属于软件工程过程，而需求分析过程在软件工程过程之前。这就隐含着一个默认的先决条件：在 CMM 这个体系下，产品在进入软件测试阶段的时候，软件需求是已经明确下来并文档化了的。

实际情况却经常并非如此，同样是软件需求，软件功能需求在进入测试阶段

就已经产生了各种文档，包括需求文档和设计文档，确保功能需求是详细、明确、无二义性的；而软件性能需求往往进入了性能测试阶段还不明确（可参见 Controller 一章开篇的例子）。这会给性能测试项目带来很大的风险。

因此，我们应该突破已有的理论束缚，寻找更合适的性能测试过程模型。经过对多个性能测试项目的实践经验总结，我们在本节提出 **GAME（A）性能测试过程模型**，其开始于软件需求分析阶段，非常符合目前国内的性能测试实践。

第二，性能测试本身有没有质量？

以前我们总是讨论软件产品的质量、开发代码的质量，但对软件测试的质量却鲜有提及。我们知道“一个好的测试用例是发现了一个原先未发现的 bug”，这其实是对用例质量的度量。软件性能测试项目也有质量，并可以度量。下面是部分度量的方法：

（1）性能测试耗费的资源，包括时间、人力、物力。

（2）性能测试中发现的 bug 数目，以及各自的级别。

（3）软件系统交付用户，在生产环境运行后发现的性能 bug 数目、级别。

而一个好的性能测试过程模型对提高性能测试质量是很有帮助的。

GAME（A）性能测试过程模型：

- G：Goal，目标
- A：Analysis，分析
- M：Metrics，度量
- E：Execution，执行
- （A）：Adjust，调整。E 执行失败后才进入 A 阶段，并且涉及的大多是有关开发和系统管理工作，因此 A 设为隐式。

性能测试过程模型如图 1-5 所示。

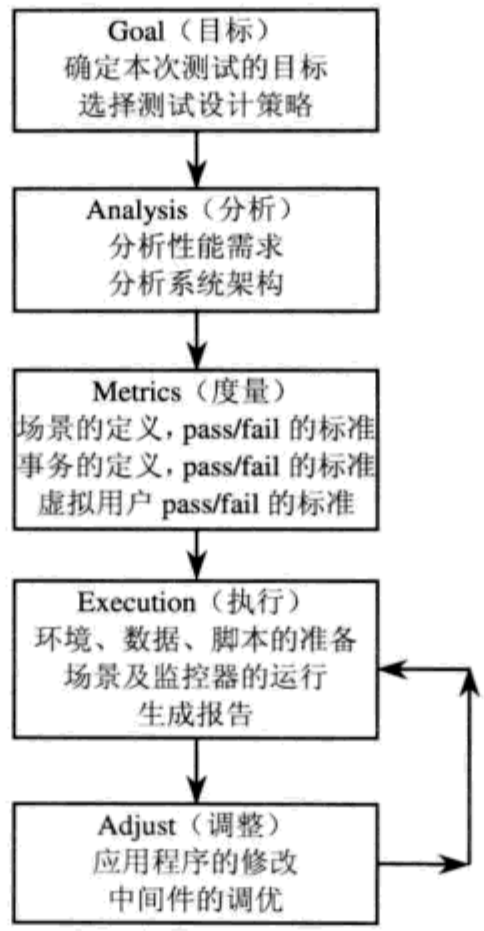


图 1-5 性能测试 GAME(A)模型

1.3.1 Goal（定义目标）

制定一个明确而详细的测试目标是性能测试开始的第一步，也是性能测试成功的关键。

本步骤的开始时间：需求获取阶段

本步骤的输入：性能需求意向

本步骤的输出：明确的性能测试目标和性能测试策略

常规的性能测试目标有以下几种：

（1）度量最终用户响应时间

查看用户执行业务流程以及从服务器得到响应所花费的时间。例如，假设

我们想要检测：系统在正常的负载情况下运行时，最终用户能否在 20 秒内得到所有请求的响应。图 1-6 显示了一个银行应用程序的负载和响应时间度量之间的关系。

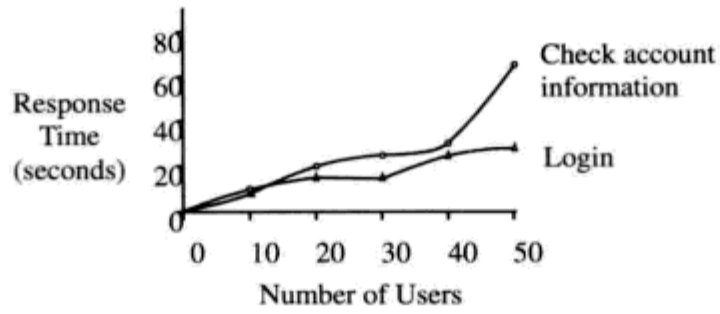


图 1-6　用户数-响应时间关系图

（2）定义最优的硬件配置

检测各项系统配置（内存、CPU 速度、缓存、适配器、调制解调器）对性能的影响。了解系统体系结构并测试了应用程序响应时间后，您可以度量不同系统配置下的应用程序响应时间，从而确定哪一种设置能够提供理想的性能级别。

例如，您可以设置三种不同的服务器配置，并针对各个配置运行相同的测试，以确定性能上的差异：

- 配置 1：1.2GHz、1GB RAM
- 配置 2：1.2GHz、2GB RAM
- 配置 3：2.4GHz、1GB RAM

（3）检查可靠性

确定系统在连续的高工作负载下的稳定性级别。强制系统在短时间内处理大量任务，以模拟系统在数周或数月的时间内通常会遇到的活动类型。

（4）查看硬件或软件升级

执行回归测试，以便对新旧版本的硬件或软件进行比较。您可以查看软件或硬件升级对响应时间（基准）和可靠性的影响。注意：此回归测试的目的不是验证升级版的新功能，而是查看新版本的效率和可靠性是否与旧版本相同。

（5）确定瓶颈

您可以运行测试以确定系统的瓶颈，并确定哪些因素导致性能下降，例如，文件锁定、资源争用和网络过载。将 LoadRunner 与新的网络和计算机监视工具结合使用以生成负载，并度量系统中不同点的性能，最终找出瓶颈所在的位置。

（6）度量系统容量

度量系统容量，并确定系统在不降低性能的前提下能提供多少额外容量。如图 1-7 所示，要查看容量，您可以查看现有系统中性能与负载间的关系，并确定出现响应时间显著延长的位置。该处通常称为响应时间曲线的“拐点”。

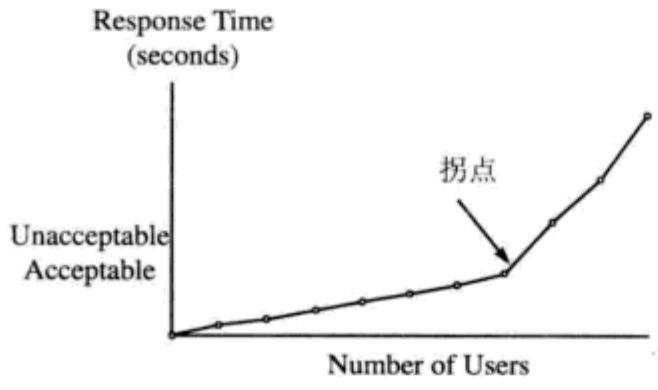


图 1-7　用户数-响应时间拐点图

我们根据不同的测试目标去选择合适的性能测试设计策略。比如，“度量最终用户响应时间”可以采用负载测试策略，“检查可靠性”就可以用压力测试策略，等等。

1.3.2　Analysis（分析）

本步骤的开始时间：需求分析阶段和性能测试启动阶段

本步骤的输入：性能需求

本步骤的输出：达成一致的性能指标列表，性能测试案例文档

1．分析性能需求

在这里，要定义性能测试的内容，细化性能需求。

客户、需求分析人员和测试工程师一起起草一个性能需求标准，对此标准获得一致认同。此标准将用户的需求细化、量化，并能在测试中作为判断依据。

比如，对于负载测试来说，可以从以下角度来细化需求，逐步找出测试关键点。

测试的对象是什么，例如“被测系统中有负载压力需求的功能点包括哪些？”；“测试中需要模拟哪些部门用户产生的负载压力？”等问题。

系统配置如何，例如“预计有多少用户并发访问？”；“用户客户端的配置如何？”；“使用什么样的数据库”；“服务器怎样和客户端通信？”。

应用系统的使用模式是什么，例如“使用在什么时间达到高峰期？”；“用户使用该系统是采用 B/S 运行模式吗？”；“网络设备的吞吐能力如何，每个环节承受多少并发用户？”等问题。

最后得出的性能测试指标标准至少要包含测试环境、业务规则、期望响应时间等。

2. 分析系统架构

对硬件和软件组件、系统配置以及典型的使用模型有一个透彻的了解。结合性能测试指标标准，生成性能测试用例。（可参看第 10 章“进阶 LoadRunner 高手”的用例设计部分。）

1.3.3　Metrics（度量）

本步骤的开始时间：性能测试设计阶段

本步骤的输入：细化的性能指标和性能测试案例

本步骤的输出：和工具相关的场景度量、交易度量、监控器度量和虚拟用户度量等

度量是非常重要的一步，它把性能测试本身量化。这个量化的过程因测试工

具的不同而异。

（1）场景的定义，pass/fail 的标准

测试场景包含了性能测试的宏观信息，有测试环境、运行规则和监控数据等。具体可表现为历史数据记录数、虚拟用户数、虚拟用户加载方式、监控指标等。

（2）事务（Transaction）的定义，pass/fail 的标准

事务用来度量服务器的处理能力。事务定义应该从性能指标标准而来，是性能指标的具体体现。事务的定义是很重要的，不同的定义会导致不同的 TPS 结果。



使用性能测试工具执行性能测试之后，我们能看到的是 pass/fail 的用户数、pass/fail 的事务数。而这些 pass/fail 的标准应该在执行性能测试之前就应该被定义好。比如，LoadRunner 默认的 pass/fail 标准是基于协议层的，而我们要的 pass/fail 可能是业务级的，需要在业务层上进行判断来决定是 pass 还是 fail。另外，还可能由于案例的关联性引起的 pass/fail，如果两个案例之间有关联，A 脚本负责向数据库插入数据，B 脚本负责查询数据，A 要是 fail 了，导致 B 也会 fail，虽然 B 本身不一定有错。解决这个问题，一方面可以削弱脚本之间的关联性，另一方面也可以通过增强脚本的健壮性来达到。

（3）虚拟用户 pass/fail 的标准

虚拟用户是性能测试工具中的一个普遍的概念，虚拟用户负责执行性能测试脚本，在这里应该定义虚拟用户遇到何种情况，选择 fail 或 pass，即退出或通过。

1.3.4 Execution（执行）

本步骤的开始时间：软件测试执行阶段

本步骤的输入：场景、交易、虚拟用户等设置信息

本步骤的输出：测试报告

执行测试包含两个工作：

1．准备测试环境、数据和脚本

测试环境：硬件平台和软件平台。

测试数据：包括初始测试数据和测试用例数据两部分。表现为 SQL 脚本、Excel 文件等。



提示　测试环境直接影响测试效果，所有的测试结果都是在一定软硬件环境约束下的结果，测试环境不同，测试结果可能会有所不同。需要注意：如果是完全真实的应用运行环境，要尽可能降低测试对现有业务的影响；如果是建立近似的真实环境，要首先达到服务器、数据库以及中间件的真实，并且要具备一定的数据量，客户端可以次要考虑。实施负载压力测试时，需要运行系统相关业务，这时需要一些数据支持才可运行业务，这部分数据即为初始测试数据。有时为了模拟不同的虚拟用户的真实负载，需要将一部分业务数据参数化，这部分数据为测试用例数据。

测试脚本：用性能测试工具生成脚本。

2．运行场景和监控性能

运行性能测试场景，并监控设定好的数据指标，最终生成测试报告。按照定义好的场景 pass/fail 标准来判断性能测试是否通过。如果未能通过，进入步骤 5（Adjust）。

1.3.5　Adjust（调整）

本步骤的开始时间：第一轮性能测试结束后，而且没有通过的条件下

本步骤的输入：测试报告和测试结果数据

本步骤的输出：性能问题解决方案

调整包含两个意思：应用程序修改和中间件调优。

中间件调优可考虑如下因素操作系统调优：

- 数据库调优；
- 内存升级；
- CPU 数量；
- 代码调优；
- Cache 调优。



解决一个性能瓶颈，往往又会出现另外的瓶颈或者其他问题，所以性能优化更加切实的目标是做到在一定范围内使系统的各项资源使用趋向合理和保持一定的平衡。

系统运行良好的时候恰恰也是各项资源达到了一个平衡体，任何一项资源的过度使用都会造成平衡体系破坏，从而造成系统负载极高或者响应迟缓。比如 CPU 过度使用会造成大量进程等待 CPU 资源，系统响应变慢，等待会造成进程数增加，进程增加又会造成内存使用增加，内存耗尽又会造成虚拟内存使用，使用虚拟内存又会造成磁盘 IO 增加和 CPU 开销增加（用于进程切换、缺页处理的 CPU 开销）。

从以上内容可以看出，目前 GAME（A）模型有两个优势：第一，灵活，每个过程都有自己的关注点，可以根据不同的项目特点增加或删除关注点；第二，通用，不依赖于具体的工具。目前 GAME（A）关注性能测试技术，比较简单，将来可以进行扩展，同样使用 GAME（A）模型关注性能测试的时间、人力等资源问题。

1.4 性能测试工具的评估和选择

我们可以看到，性能测试和一般功能测试不同的是，性能测试的执行是基本功能的重复和并发，因此我们在性能开始之前需要模拟多用户，在性能测试进行时要监控指标参数，同时性能测试的结果不是那么显而易见，需要对数据进行分

析。这些特点决定了性能测试更适合通过工具来完成。市场上涌现出越来越多的压力自动化测试工具，古人云“工欲善其事，必先利其器”，一个测试工具能否满足测试需求，能否达到令人满意的测试结果，是选择测试工具要考虑的最基本的问题。

我们这里讨论的主要是一些比较成熟的性能测试软件产品，都已经在市场上占有了一定的份额，得到了用户的认可。

如表 1-1 所示为主要的性能自动化测试工具。

表 1-1　主要的性能自动化测试工具

工具名	公司（组织）	License	描述
WAS （ Web Application Stress Tool）	Microsoft	需要	适用于 B/S 架构，模拟浏览器请求
Qaload	Compuware	需要	支持多种系统架构
LoadRunner	Mercury Interactive	需要	支持多种系统架构
Astra quick test	Mercury Interactive	需要	支持多种系统架构
OpenSTA	OPENSTA 组织	开源	支持 HTTP 协议
Jmeter	Apache	开源	全面支持 Java

对于测试人员来说，要么自己开发性能测试工具，要么选择购买市场上已有的性能测试工具。在这里，我们要讨论的只是在选择性能测试工具时，需要考虑哪些因素。这些因素都想清楚了，然后才可以做决定。

1.4.1　测试预算 VS 工具价格

性能测试的成本与收益比是选择性能测试工具的根本条件。这其实是在考虑“要不要用”的问题。如果购买一套价格几十万的性能测试工具只是为了去做一个几万元预算的性能测试项目，那么无论这个工具再强大，也不会被采用。

1.4.2　协议、开发技术、平台、中间件 VS 工具的支持

要确定性能测试工具是否支持我们的被测软件系统，这其实是在考虑“能不

能用”的问题。考虑因素有被测软件系统使用的协议、采用的技术、基于的平台、调用的中间件。这些都需要性能测试工具有效的支持。

1.4.3 工具可使用的复杂程度 VS 项目计划的影响

熟悉并使用一个性能测试工具，是需要花费人力和时间等资源的，项目计划中要有相应的资源准备。这其实是在弄清“如何用”的问题。

2

第2章

第一次亲密接触LoadRunner

LoadRunner 是 Mercury Interactive 公司开发的一款成熟的性能测试工具，LoadRunner 作为性能测试的实现者，涉及了性能测试流程、性能测试技术和软件体系架构等众多方面的知识点，可以说，学习 LoadRunner 是理解和学习性能测试的非常好的切入点。在本章，我们将从架构上了解一下 LoadRunner 的三大模块，看看它们是如何共同协作完成一次软件性能测试的。

2.1 从性能测试到 LoadRunner 的映射

可以想象，如果没有性能测试工具，那么我们靠手工将如何做性能测试：

首先，我们需要一个测试执行指挥官和一群测试人员，测试指挥官负责指挥和协调整个测试过程，而测试人员则按照指挥官的口令一起去操作自己的终端。准备好之后，指挥官通知大家“开始”，然后各个测试人员开始按照测试案例的

步骤来执行操作，并把自己的操作时间记录下来，在性能测试完毕后，记录下来汇报给测试指挥官，测试指挥官把结果数据进行计算，得出性能测试是否通过的结果。

显然，这样做应该是最符合软件系统的实际使用情况的，但是实际上我们在做性能测试的时候，我们必须要提前考虑几个问题：

（1）资源。显然，手工性能测试需要多个测试人员和多台终端设备，如果性能需求中说明软件系统的用户可能来自不同网络配置，如局域网、ADSL、拨号等，那么做性能测试还要考虑制造这些条件，这无疑会增加测试的成本。

（2）协调。指挥官需协调各个测试人员，让他们同时开始，如果出现了问题，测试指挥官应该立即决断，是继续测试，还是立即中断，去解决问题。

（3）重复。性能测试不可能一蹴而就，需要多轮测试，那么如何在相同的条件下重复第一轮测试，手工测试必然会带来误差。

（4）分析。测试指挥官需要分析结果数据，得出性能测试结果是否通过，如果没有通过，最好还要指出是软件系统的哪个节点出了问题，提供解决问题的线索。

我们来看看 MI 的 LoadRunner 是怎样解决这些问题的，如图 2-1 所示。

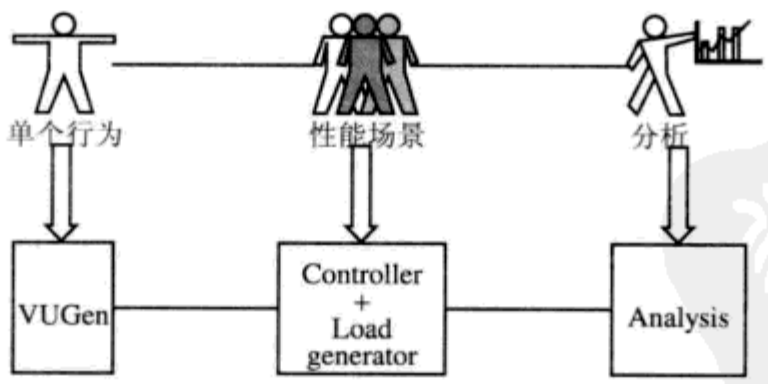


图 2-1　性能测试到 LoadRunner 的映射

（1）测试人员被 LoadRunner 的 Vuser 代替（虚拟用户），测试人员执行的操作以 Vuser Script（虚拟用户脚本）的方式固化下来。而一台计算机上可以运行多个 Vuser，因此 LoadRunner 又减少了性能测试对硬件的要求。



虚拟用户：

虚拟用户是性能测试自动化实现中一个普遍的概念，顾名思义，虚拟用户并不是一个个真实的人，但对被测试的服务器来说，这些虚拟用户和真实用户的行为并无差别。可以说，被测服务器并不清楚哪些请求是来自真实的人还是虚拟用户。在 LoadRunner 中，虚拟用户的实现原理是 VU 脚本＋run-time setting。

虚拟用户的另外一个好处是，一台工作站只能容纳一个实际用户，而多个 Vuser 可以同时以进程/线程的形式运行在同一台工作站上。实际上，一个测试场景可以包含几十、几百甚至几千个 Vuser。

虚拟用户脚本：

Vuser 在方案中执行的操作是用 Vuser 脚本描述的。运行场景时，每个 Vuser 都去执行 Vuser 脚本。Vuser 脚本记录了用户的动作，并且包含一系列度量并记录服务器性能的函数，从而方便计算性能指标。这就像一个真实的用户一边做操作，一边拿着秒表记录时间一样。

（2）测试指挥官被 LoadRunner 的 Controller 替代，顾名思义，Controller 就是运行性能测试的司令部，Controller 负责生成性能测试场景，管理和协调多个虚拟用户，在实际运行时，Controller 运行任务分派给各个 Load generator，同时还联机监测软件系统各个节点的性能，并收集结果数据，提供给 LoadRunner 的 Analysis。



性能测试场景：

使用 LoadRunner 可以将应用程序性能测试需求划分为多个场景。场景定义并控制要模拟的用户的数量及其执行的操作，以及他们运行模拟操作所用的计算机。简单说来，场景是把虚拟用户按一定规则组织起来去模拟真实世界的业务行为。场景的组织通常和真实世界的业务规则有很大关系。

（3）Analysis 会对数据进行整合，分析出它们之间的关系，并把这些关系以图表和报告的形式展现出来，使得性能测试的结果一目了然。



Analysis：

Analysis，名副其实，就是 LoadRunner 用来分析数据的工具。一个测试场景运行结束后，会在指定的目录下生成一个 Access 数据文件，这时启动 Analysis，就可以分析这些数据了。

这些概念之间的关系，可以用下面的比喻来形象地说明。

把性能测试比作一场盛大的演出，虚拟用户（Vuser）就是演员，导演（Controller）则规定了参加演出的有多少个演员，各自如何在舞台上表演，这种规则就是剧本（性能测试场景）。导演要在各个方面保证剧情不能离谱、脱离现实，否则就要砸锅了。因此，导演不仅要确保演出能顺利结束，而且还要同时观察和收集观众的反馈信息（联机监测功能），以分析（Analysis）、确认这次演出是否成功（性能需求是否到达）。

2.2 LoadRunner 工作原理

现实中，性能测试这场演出就是这么进行的，如图 2-2 所示。

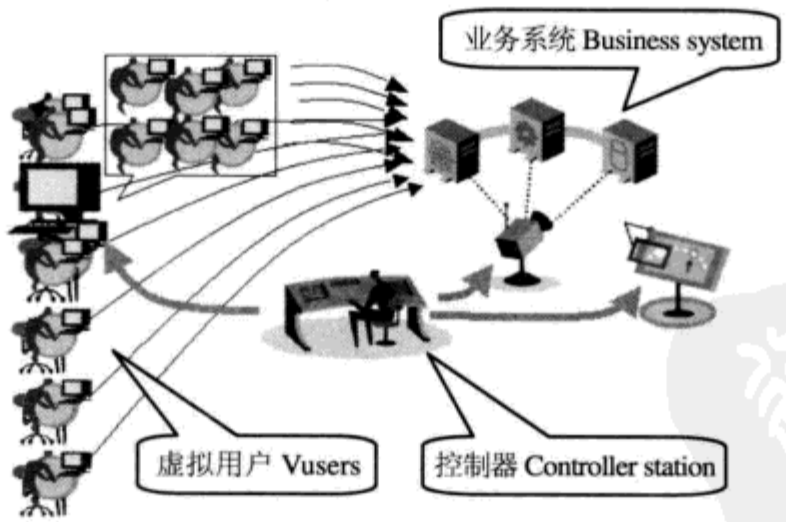


图 2-2 LoadRunner 工作原理流程图

（1）性能测试人员操纵运行 Controller，启动性能测试。

（2）大量的虚拟用户被“激活”，和真实用户所做的一样，向被测系统发送

请求。

（3）Controller 通过 monitor 能够实时捕获系统所有层面，包括服务器、网络资源的性能数据，并显示在 Controller 上。

（4）执行结果存放在数据库里，用户通过 Analysis 生成报告并进行测试结果分析。

实际上，LoadRunner 为了保证这场“浩大演出”的成功，除了三个模块的基本功能之外，还提供了一些“细心而方便”的功能，这使得 LoadRunner 具有更好的易用性。

1．Vuser 提供录制/回放功能

学习 LoadRunner 很容易上手，其关键是 LoadRunner 的 Virtual User Generator 具有录制/回放功能。可以通过录制直接生成脚本，极大地节省了开发和调试脚本的工作量。VU 可以先记录下业务流程，然后将其转化为测试脚本。建立测试脚本后，可以对其进行参数化操作，这样可以利用几套不同的实际发生数据来测试应用程序。以一个订单输入过程为例，参数化操作可将记录中的固定数据，如订单号和客户名称，由可变值来代替。在这些变量内随意输入可能的订单号和客户名，来匹配多个实际用户的操作行为。

2．Controller 机智地创建性能测试方案

虚拟用户脚本生成后，使用 LoadRunner 的 Controller 可以根据需要设定负载方案、业务流程组合和虚拟用户数，创建不同的性能场景。

Controller 能够创建基于目标和手工的性能测试场景，前者可使用户关注于某项他感兴趣的性能指标，后者则可全面考察整个软件系统在既定负载下的性能表现。实际上，我们可以通过手工和目标场景的设置来实现在上一章提到的性能测试策略，如表 2-1 所示。

表 2-1 LoadRunner 场景对性能测试策略的映射

性能测试策略	LoadRunner 性能测试场景
压力测试	面向目标测试场景＋忽略 think time
负载测试	手工测试场景＋同步点＋think time＋虚拟 IP＋带宽模拟……
基准测试	脚本和场景复用
并发测试	同步点＋多虚拟用户

（有关 think time、同步点、虚拟 IP 等概念，将在后续章节中为大家详解。）

3. 数据驱动

LoadRunner 做到了将业务流程和业务数据分离，并且很彻底。其中有一个很有用的功能，就是 Data Wizard。通过它的 Data Wizard 来自动实现其测试数据的参数化。Data Wizard 直接连于数据库服务器，从中可以获取所需的数据（如订单号和用户名）并直接将其输入到测试脚本。这样避免了人工处理数据的需要。

4. 服务器监测

LoadRunner 可检查出现性能延迟的地方：网络或客户端延迟、CPU 性能、I/O 延迟、数据库锁定和数据库服务器上的其他问题。这是因为 LoadRunner 内含集成的实时监测器，在负载测试过程的任何时候，都可以观察到应用系统的运行性能，包括服务器、数据库、网络设备等。

5. 测试结果分析

一旦测试完毕后，LoadRunner 收集汇总所有的测试数据，可以提供高级的分析和报告工具，以便迅速查找到性能问题并追溯原由。使用 LoadRunner 的 Web 交易细节监测器可以了解到将所有的图像、框架和文本下载到每一个网页上所需的时间。例如，这个交易细节分析机制能够分析是否因为一个大尺寸的图形文件或是第三方的数据组件造成应用系统运行速度减慢。另外，Web 交易细节监测器分解用于客户端、网络和服务器上端到端的反应时间，便于确认问题，定位查找真正出错的组件。例如可以将网络延时进行分解，以判断 DNS 解析时间，连接服

务器或 SSL 认证所花费的时间。通过使用 LoadRunner 的分析工具，能很快地查找到出错的位置和原因并做出相应的调整。

以上这些功能我们都将在后面的章节中详细地介绍。

2.3　安装 LoadRunner

目前 MI 的 LoadRunner 提供了 Windows 版本和 UNIX 版本，也就是说，LoadRunner 可运行在 Windows 平台和 UNIX 平台下。这取决于被测软件系统对客户端的要求。

而安装 LoadRunner 对计算机配置有要求，如表 2-2 所示。

表 2-2　LoadRunner 安装系统要求

系统部件	Controller 和实时监视器	虚拟用户生成器	虚拟用户	Analysis 模块
计算机/处理器	Pentium 350MHz 以上	Pentium 350MHz 以上	Pentium 1GHz 以上	Pentium 350MHz 以上
操作系统	Windows NT 4.0 SP6 以上，Windows 2000 SP2、SP3，Windows XP Professional SP1	Windows NT4.0 SP6 以上，Windows 2000 SP2、SP3，Windows XP Professional SP1	Windows NT4.0 SP6 以上，Windows 2000 SP2、SP3，Windows XP Professional SP1	Windows NT 4.0 SP6 以下，Windows 2000 SP2、SP3，Windows XP Professional SP1
内存	128MB 以上	128MB 以上	未采用多线程的 Vuser 要求 RAM 达到 1MB 以上。采用多线程的 Vuser 要求 512KB 以上	128MB 以上
交换空间	物理内存总量的 2 倍	物理内存总量的 2 倍	物理内存总量的 2 倍	物理内存总量的 2 倍
硬盘空间	安装：300MB	安装：300MB	安装：130MB	安装：100MB
	可用空间：200MB	可用空间：200MB	可用空间：最小 500MB	可用空间：最小 500MB
浏览器	IE 5.x 及更高版本，Netscape Navigator 4.x、6.x	IE 5.x 及更高版本，Netscape Navigator 4.x、6.x	N/A	IE 5.x 及更高版本，Netscape Navigator 4.x、6.x

随着计算机硬件成本的降低，上面的系统要求已经很容易达到。但是需要注意的是，它们是最小的资源要求，我们在安装 LoadRunner 的时候，往往要把这个硬件要求提高一些，以保证性能测试的正常执行。具体安装步骤在本书就不花篇幅介绍了，可参看 LoadRunner 安装手册。

2.4 揭开 License 的神秘面纱

License 又叫做授权，使用授权（License）是版权的拥有者对作品的使用者行为的一种文档化的约束和规定。使用授权一般规定了用户使用该作品的条件，是否能够修改或者重新发布。从法律意义上来讲，License 是一种契约或者合同，具有法律效力。软件用户通过购买 License 的方式来和软件厂商达成契约关系。

一般地，软件的 License 的实现方式分为硬件实现和软件实现两种。硬件实现是将软件与硬件绑定，通过硬件来实现对用户鉴权，代表性的做法有加密狗等；License 的软件实现则有序列号等。

目前，LoadRunner 的 License 实现方式硬件和软件都有，应用在不同的功能模块中。

可以通过 LoadRunner Controller 来查看 License 信息。

在 Controller 中，选择“帮助”＞“关于 LoadRunner”，查看 Controller 的版本信息。打开“LoadRunner Controller 版本信息”对话框，单击“License Information”按钮，即可显示许可证信息，如图 2-3 所示。

在时间上，LoadRunner 有以下可用许可证类型：

- Permanent：这一类型的许可证没有有效期限限制。
- Time limited：这一类型的许可证有期限限制。
- Temproray：这一类型的许可证只能保证自产品安装后 10 天的使用期。

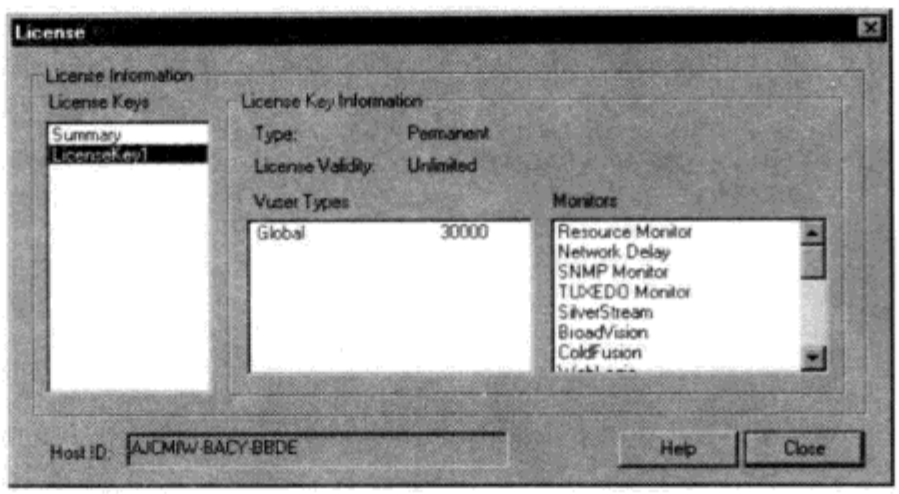


图 2-3　LoadRunner License 信息

- VUD-based：这一类型的许可证受虚拟用户天数（VUD）的限制。在 24 小时内，VUD 许可证允许用户使用该产品无限次。
- Plugged：这一类型的许可证需要加密狗。

在空间上，LoadRunner 对可用虚拟用户类型也有限制，有以下内容：

- Global：Global 许可证意味着只要不超过 Vuser 总数限制值，可以运行任意类型的 Vuser。Global 后面跟着的数字就是允许的 Vuser 总数。
- Group ERP-1：包含 SAP、Web、Oracle、MSSQL 和 Siebel（GUI-Oracle）虚拟用户协议。
- Group ERP-2：包含 SAP、Siebel（GUI-Oracle）、Oracle、Oracle-NCA 虚拟用户协议。
- Group3：包含除 SAP、Siebel（GUI-Oracle）、Oracle、Oracle-NCA 以外的所有虚拟用户协议。

有以下可用监视器类型：系统资源、Web 资源、web 服务器资源、Web 应用程序服务器资源、数据库服务器资源、网络延迟、流媒体资源、防火墙服务器资源及 ERP 服务器资源。

当然包含越多功能的 License 就越贵，我们使用 LoadRunner 时，应该根据项目需求，选择合适的 License。



第3章

走近LoadRunner

如果我们留心和品味身边的事物，总会发现一个规则：强大的事物一般既是简单的，又是复杂的。简单指的是我们对事物的感受，而复杂指的是事物背后的原理和规律。比如阳光，阳光是地球万物的生长之源，我们无时无刻不直接或间接地感受它的存在，但对于光本身，太复杂了，以至于从古到今对光的原理争论不休，有微粒学、波动学和量子论的各种解释方法。同样的，对于软件来说也是如此，一个强大的软件应该既是简单的，又是复杂的，简单的是软件用户的感受，而复杂的则是软件背后的设计框架和实现。

上一章我们讲到了 LoadRunner 简单的一面，我们在这章就来看看 LoadRunner 复杂的一面，了解 LoadRunner 的工作原理会让我们更加有效地使用 LoadRunner。

3.1　LoadRunner 的运行原理

安装 LoadRunner 后，在菜单“开始”→“Mercury LoadRunner”中，你会看到这样一组程序，如图 3-1 所示。

其中 Applications 下面的 Analysis、Controller 和 Virtual User Generator 是我们做性能测试最常用的三个工具。



图 3-1　Windows“开始”菜单中的 LoadRunner 程序组

3.1.1　LoadRunner 三大高手

点击 Virtual User Generator，VU 就会被启动，我们会看到这样一个窗口界面（LoadRunner 8.0），如图 3-2 所示。

启动 VU 后，我们会在任务管理器中看到多了一个 Vugen.exe 进程。虚拟用户产生器 Vugen 实际上是一套开发调试脚本的环境，它的任务是生成编译/解释成功的脚本，供 Controller 调用。注意：虽然都叫做 VU Script，但其实不同的 Vuser 类型却暗地里“各自为政”，有的是解释执行的真脚本，比如 C Vuser；有的是编译执行的“伪脚本”，如 Java Vuser 脚本。它们都将有一定的接口与 Controller 对

接，以便 Controller 调用。

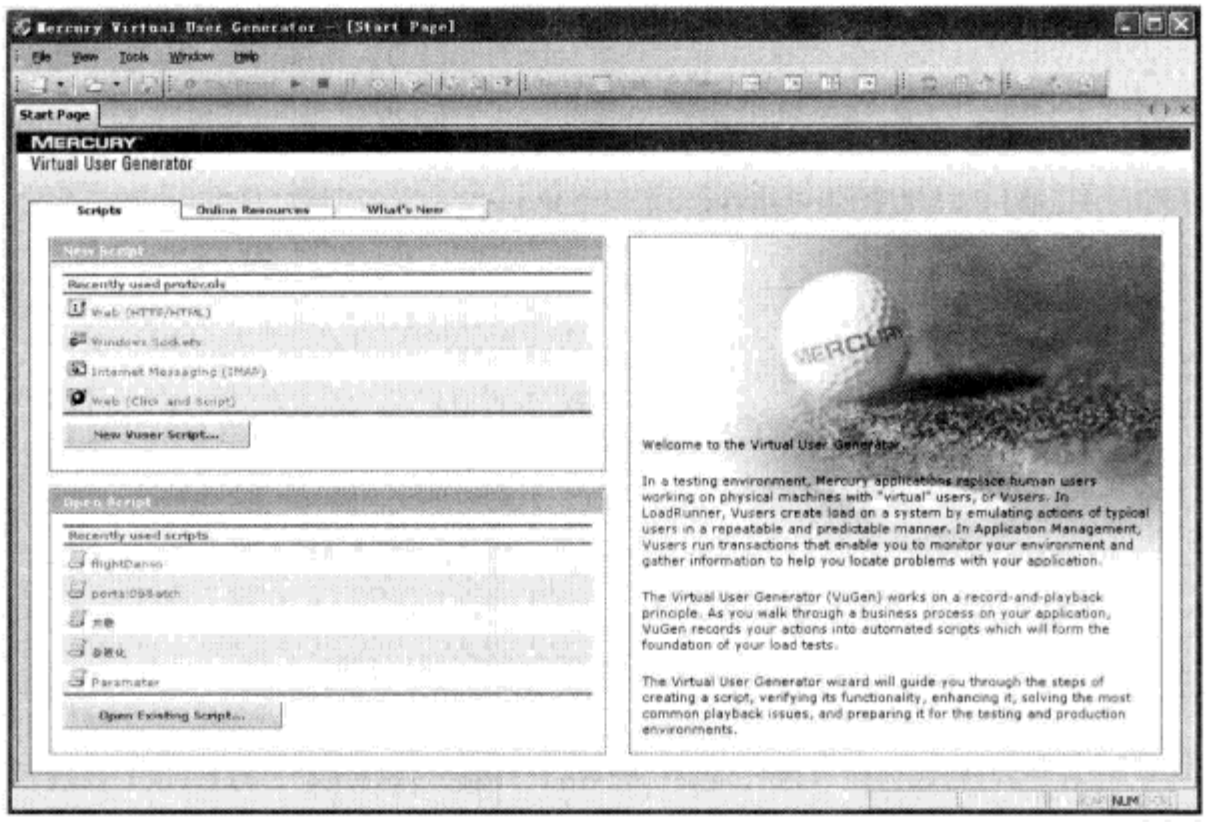


图 3-2 VU 主界面

现在我们选择 Web（http/html）协议，新建一个空白脚本，然后保存成一个名为“example”的脚本在 C 盘下，打开 C 盘，我们会看到在 C 盘下生成了一个“example”的目录，没错，脚本其实就是一个目录，所有该脚本的信息都将放在这个目录下，如图 3-3 所示，我们一一解读。

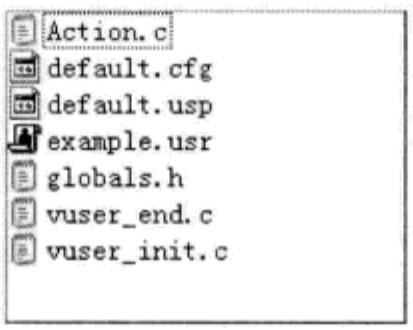


图 3-3 VU 脚本目录文件

首先有三个 c 文件，分别是 Action.c、vuser_init.c 和 vuser_end.c，就是我们在 VU 编辑器中看到的 action.c、vuser_init、vuser_end 函数。如果是 Java Vuser，则只有一个 action.java 文件。

Default.cfg：是脚本运行的配置文件。通过 VU 界面设置的 run-time 等信息存储在这个文件里，同样我们也可以直接修改这个文件而达到设置 run-time 的目的。这其实为我们的性能测试驱动自动化提供了接口。

如果脚本经过解释/编译后，还会生成以下几个文件：

Pre_cii.c：打开这个文件，里面藏着很多宝贝，VU 的“后台支持”都在这里呢，可以看到，虽然我们在 VU 编辑器中只有三行代码，但生成的最终 C 文件已经达到了几百行，包括 LoadRunner 引用的各个程序文件和外部函数。“前端简单，后台复杂”的道理再次被验证。而 Java 脚本则编译后直接生成了一个 action.class 文件。

脚本运行的同时，还会生成 output.txt 和 mdrv.log 等文件，这些文件记载了 VU 运行的 log，如果我们在碰到一个无从下手的奇怪错误，可以到这些文件里试试运气。

下面，我们再来看看 Controller。启动 Controller，我们看到如图 3-4 所示的界面。

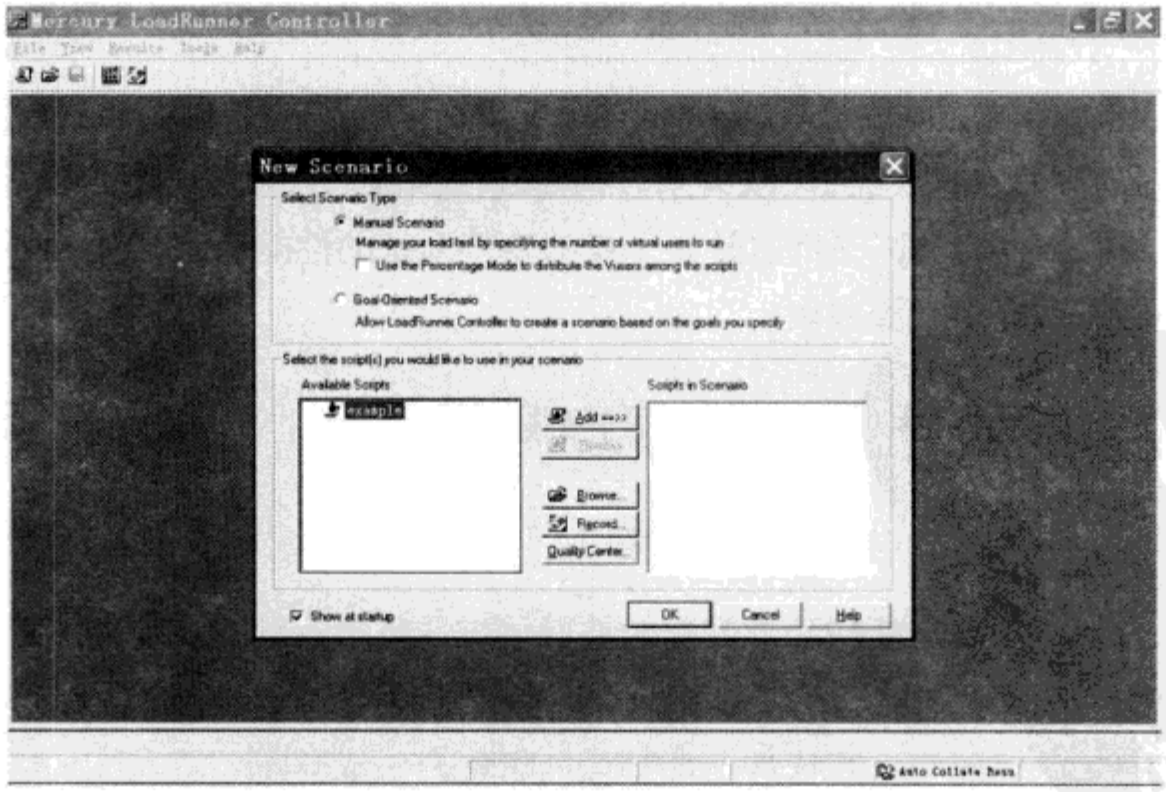


图 3-4　Controller 启动界面

Controller 是整个性能测试的核心，它是一个框架程序，与 LoadRunner 的各个模块都有交互，可以说 Controller 是“集大权于一身”，控制着整个性能测试的过程。第一，定义目标，Controller 将 Vugen 开发的脚本按一定规则组织成性能测试场景，其中有面向目标场景和手工场景（如图中的 Maunaul Scenario 和 Goal-Oriented Scenario）。第二，分派执行，Controller 规划测试拓扑，把测试任务指定在若干个 Load Generator 上以多进程/多线程的方式在运行。第三，度量评估，Controller 同时实现无代理方式的监控功能，监控多种服务器的多个性能参数，并把监控的结果数据提供给 Analysis 进行分析，后者得出此轮测试是否通过的结论。



“定义”、“执行”和“评估”是科学做事的三个基本而关键的要素，在我们的社会生活中其实处处都有体现，小到一个软件项目，大到一个国家政体。比如立法、行政、司法三权分立思想，为什么不是五权分立或七权分立？因为三权恰恰对应了这三个要素，立法对应“定义”，行政对应“执行”，司法对应“评估”。很显然 LoadRunner 的设计者考虑到了这些要素同样存在于性能测试中，并通过 Controller 来实现。

而 Analysis 实际上是 LoadRunner 里最“聪明”的模块。它负责在一大堆数据里寻找不同数据之间的关联关系，构建线性函数，以报表、图示的方式展现给用户。我们做性能测试的目的是为了找到软件系统的性能瓶颈，而 Anaylysis 会给我们分析瓶颈提供重要线索。

3.1.2 三大高手联手的一场性能测试盛大演出

好，三大高手已经亮相了，我们下面开始介绍它们在性能测试中的演出。

（1）首先确定哪些功能需要做性能测试，然后启动 VU 录制相应的业务，生成自动化脚本。在这个录制过程中，VU 就是一个超强模仿秀，用户所做的每一个操作，都暗记在心，并认真记录笔记，也就是生成脚本。

（2）如果说性能测试的总导演是性能测试工程师，那么 Controller 就是当之无愧的执行导演了。Controller 首先把演员（虚拟用户）都召集齐，然后告诉把剧本（场景）交给它们（VU 脚本），分配到各自的舞台（Load Generator），一声令下，演出开始！



注 意　当 Controller 运行场景任务时，在任务管理器里可以找到一个名为“lr_bridge”的进程，顾名思义，它就是起到联系 Controller 和 Generator 的桥梁作用，Controller 的指令就是通过它来下达到 Load Generator 的。而 Load Generator 接到运行指令后，会启动一个 mdrv.exe，以多进程/多线程方式调用运行脚本文件。通常一个 mdrv.exe 可以驱动 50 个线程或一个进程。如果在测试场景中定义更大的 Vuser 数目，那就要开启更多的 mdrv。

（3）执行导演 Controller 在开机之后，它最关心的是演出效果是否达到预期，于是它在演出的同时，还在现场搜集台下的观众对这场演出的评价。其中最重要的就是各个 Load Generator 记录的最终用户响应时间和脚本执行的日志，这是演出是否达到预期最直接的判断依据。

（4）演出终于结束了，Controller 执行导演命令各个 Load Generator 把它们记录的数据传送回 Controller，然后 Controller 执行负责将数据汇总，存储在结果目录下的 output.mdb 文件里。这其实是一个 Access 数据库，如果我们打开这个数据库会发现有多个数据表，存储着不同类型的数据。

（5）在没有保存之前，结果数据是存放在 Windows 临时目录下的。保存之后，数据分析工具 Analysis 从 output.mdb 中读取测试结果数据，进行分析工作，展现出最后这场盛大演出的总结报告，而我们的性能测试工程师可以从这份报告里发现蛛丝马迹，确定瓶颈，找出关键的调优因素。



多进程/多线程方式：

用户可以在 Controller 的 run time setting 中选择 Vuser 的运行方式：多线程/多进程。多进程和多线程方式的区别是：

Controller 将使用驱动程序 mdrv 运行 Vuser。如果按进程方式运行每个 Vuser，则对于每个 Vuser 实例，都将启动一个 mdrv 进程。多个 mdrv 进程会占用大量内存及其他系统资源，这就限制了可以在任一负载生成器上运行的 Vuser 的数量。如果选择按线程方式运行，在默认情况下，Controller 为每 50 个 Vuser 仅启动一个 mdrv.exe 进程，而每个 Vuser 都按线程运行，这些线程 Vuser 将共享父进程的内存段。这就节省了大量内存空间，从而可以在一个负载生成器上运行更多的 Vuser。



但任何选择都是有两面性的。选择线程方式运行 Vuser 会带来一些安全问题。因为线程的资源是从进程资源中分配出来的，因此同一个进程中的多个线程会有共享的内。存空间，这样可能会引起多个线程的同步问题，调度不好，就会出问题，比如 A 线程要用的资源必须等待 B 线程释放，而 B 也在等待其他资源释放才能继续。这就是有些网友碰见的问题：同一个测试场景，用线程并发就会超时失败或报错，而用进程并发就没错。



在 LoadRunner 中，只有线程安全协议才能作为线程运行。需要注意的是下列协议不是线程安全协议：Sybase-Ctlib、Sybase-Dblib、Informix、Tuxedo 和 PeopleSoft-Tuxedo。

3.2 LoadRunner 的录制原理

3.2.1 网络协议与 LoadRunner 的 Vuser

LoadRunner 提供了多种 Vuser 技术，通过这些 Vuser 技术，LoadRunner 可以

在不同类型的客户端/服务器体系结构下生成相应的脚本。

我们在新建一个 VU 脚本时，会弹出这样一个选择对话框页面，如图 3-5 所示。

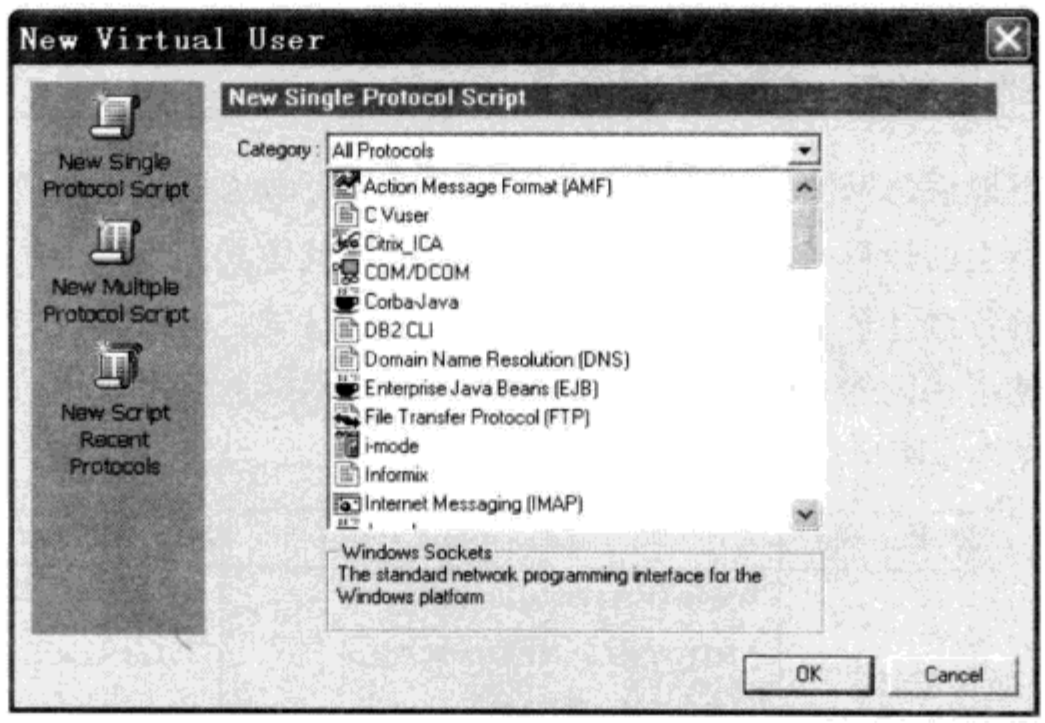


图 3-5　VU 协议选择列表

在全部协议列表（All Protocols）里，有几十种协议供选择，相信刚开始接触 LoadRunner 的朋友，看到这么多的名词，必然心生许多疑惑：

（1）VU 提供的 Protocol 是干什么用的？也就是说，在录制脚本时，为什么要选择 Protocol 类型？

（2）被测软件系统采用的协议与 LR 提供的 Protocol 类型的关系？比如 VU 里的 SMTP Protocol 和邮件系统的 SMTP 邮件发送协议有什么样的关系？

（3）在对一个已知协议的软件系统做性能测试时，我们有什么技巧或原则去选择 VU Protocol 类型么？

在得到这些问题答案之前，我们有必要再次翻出计算机网络教材，温习一下网络知识。

网络协议：

网络协议是网络通信技术与计算机技术相结合的产物。所谓计算机网络，就是把分布在不同地理区域的计算机与专门的外部设备用通信线路互联成一个规模大、功能强的网络系统，从而使众多的计算机可以方便地互相传递信息，共享硬件、软件、数据信息等资源。

通俗地说，网络就是通过电缆、电话线或无线通信等互联的计算机的集合。那么网络上的计算机又是如何交换信息的呢？就像我们说话用某种语言一样，在网络上的各台计算机之间也有一种“语言”，这就是网络协议，不同的计算机之间必须使用相同的“语言”，才能进行通信。网络协议可以用一个层状模型来描述，如图 3-6 所示。

	标准开放协议	中间件协议
应用层	DNS,FTP,HTTP,IMAP,IIRC, NNTP,POP3,SMTP,SNMP,SSH, LDAP,Telnet,BitTorrent, DHCP…	Tuxedo,oracle, sqlserver,corba
传输层	DCCP,SCTP,TCP,RTP,UDP,IL,RUDP…	
网络层	IPv4,IPv6...	
数据链路层	以太网,Wi-Fi,令牌环,MPLS,PPP…	
物理层	RS-232,EIA-422,RS-449,EIA-485,10BASE2, 10Base-T...	

图 3-6　网络协议结构

各层的关系是，上层协议的功能都是在其下层协议功能的基础上实现的。从物理层的原始信号处理开始，每一层协议完成自己的功能，下层协议为上层协议提供服务，直到应用层协议得到数据，向上提供给网络应用。层次越低的协议就越接近机器语言，也就越难懂；层次越高的协议就越接近人类语言，也就越好理解。

通常，网络层是一个分水岭，网络层向上是有关计算机软件技术，网络层向下是有关计算机通信技术。因此软件人员能接触到的通常是在应用层和传输层。应用层协议种类繁多，层出不穷，常用的网络应用协议如表 3-1 所示。

表 3-1　常用的网络应用协议

网络协议名称	解释	构建的网络应用
Telnet（Teletype over the Network，网络电传）	通过一个终端（Terminal）登录到网络（运行在 TCP 协议上）	网络应用有基于 Telnet 的 BBS 系统
HTTP（HyperText Transfer Protocol，超文本传输协议）	是互联网上应用最为广泛的一种网络传输协议。所有的 WWW 文件都必须遵守这个标准。HTTP 协议是目前应用最广泛的应用层协议	所有 WWW 的网络应用都是基于 HTTP 协议
FTP（File Transfer Protocol，文件传输协议）	用于文件传输，运行在 TCP 协议上	FTP 客户端
SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）	用来发送电子邮件（运行在 TCP 协议上）	常见的 SMTP 协议应用就是邮件系统
DNS（Domain Name Service，域名服务）	用于完成地址查找，邮件转发等工作（运行在 TCP 和 UDP 协议上）	DNS 服务解析
SNMP（Simple Network Management Protocol，简单网络管理协议）	用于网络信息的收集和网络管理	网管软件

除了这些标准开放协议外，还有一些常见的中间件，如 oracle、sqlserver、Tuxedo 等，它们的客户端和服务器通信的方式，也可看做是应用协议的一种。

LoadRunner Protocol 对网络协议的映射

在 VU 新建一个脚本的时候，LoadRunner 会提示让我们选择协议。选择并进入协议环境中，才能录制和编辑脚本。同时，LoadRunner 的手册会告诉我们 LoadRunner 的协议类型有 corba，Com，DB2，DNS，EJB，FTP，i-mode，SQL Server，Oracle，Tuxedo，SMTP，HTTP，Winsocket 等多种类型。但手册并没有告诉我们 LoadRunner 为什么会有不同的协议类型？为什么是这些 Vuser，而不是其他的？

图 3-7 说明了 LoadRunner Vuser 和网络协议的关系。

由此可见，LoadRunner 的每个 Vuser 类型对应一种网络协议结构模型。如 FTP Vuser 对应的是 FTP 协议 http Vuser 对应的是 HTTP 协议，而 Tuxedo Vuser 对应的是 Tuxedo 客户端/服务器结构。而在计算机网络产业迅速发展的今天，网络协议架构也是“后浪推前浪”，层出不穷的。LoadRunner 则能很快地增加新

的 Vuser 来适应新的网络协议，比如，在 LoadRunner 8.0 之前没有 Web Service 的 Vuser，由于 Web Service 系统架构的普及，LoadRunner 从 8.0 开始增加了 Web Service Vuser。

	标准开放协议	中间件协议	LoadRunner Vuser
应用层	DNS,FTP,HTTP,IMAP,IIRC, NNTP,POP3,SMTP,SNMP,SSH, LDAP,Telnet,BitTorrent, DHCP…	Tuxedo,oracle, sqlserver,corba	Tuxedo,oracle, MMS,corba,FTP, http,SMTP
传输层	DCCP,SCTP,TCP,RTP,UDP,IL,RUDP,…		Winsocket
网络层	IPv4,IPv6…		

图 3-7 VU Protocol 对网络协议的映射

Case **案例分析**

可以用个例子来更加通俗形象地说明这个关系：网络协议是计算机之间沟通交流的“语言”，计算机都是通过“语言”来说话的，LoadRunner 要知道计算机之间说什么，它只需派出一个翻译间谍（Vuser）监听计算机的对话就可以了，听完了，还要记下来生成脚本，这样做就 OK 了。但是很遗憾，这个世界上不是只有一门语言，不仅有汉语，还有英语、法语等几十种语言，计算机也面临这种问题，“语言”（网络协议）有上百种，不同的计算机之间用不同的语言说话，甚至方言也都有（增强协议），在这种情况下，我们可怜的 LoadRunner 只好增加人手，培养不同语言的翻译人才（LoadRunner Protocol），如果计算机是用 A 语言说话，LoadRunner 就要派出 A 语言翻译，如果计算机用 B 语言说话，LoadRunner 就要派出 B 语言翻译，这样问题就能得到解决了。但同时也要注意的是，一定不要派错了翻译（选错了 Protocol 类型），如果计算机用 A 语言，而 LoadRunner 派出 B 语言翻译，毫无悬念，LoadRunner 要为自己的错误负责，可怜的 B 翻译将会什么也听不懂，交了白卷。

LoadRunner 的 Vuser 和网络协议区别在于，一般地，网络协议都会有自己的一套操作原语，而 LoadRunner 的协议则对这些原语进行了一些封装工作，让它们

变得更加容易被理解和使用。比如 HTTP 协议有 get、post 原语，LoadRunner 的 http Vuser 则有 web_url、web_submit_data 函数相对应。无疑这样增加了脚本的可读性和可维护性。

需要注意的是，中间件协议和开放标准协议不同的是，开放标准协议是基于 TCP 协议上提供了一套操作原语，使用此协议不需要任何额外的安装，而中间件协议是以 API 形式提供给上层调用。因此 LoadRunner 在使用中间件类型协议时，需要保证本机已经安装了本中间件的客户端。

另外 LoadRunner 同时还提供了一些模板虚拟用户（Template Vuser），主要是针对不同的计算机语言，有 Java Vuser、C Vuser，VB Vuser、Java Script Vuser、VB Script Vuser。这些 Template Vuser 不针对任何特定协议，因此也没有录制的功能，适合自编写脚本。

正确地选择 LoadRunner 协议之后，我们就可以录制基于此协议的操作，并自动生成脚本了。如果错误地选择了协议，那我们的录制就一无所获。

3.2.2　选择 LoadRunner Protocol 的两大定律

在选择 LoadRunner 协议之前，我们首先要弄懂被测软件系统的架构，知道系统由哪些节点组成，每个节点之间的通信方式。然后根据以下原则来选择 LoadRunner 协议。

选择第一定律：以客户端和其直接连接的 Server 通信方式为准。

因为 LoadRunner 的 Vuser 模拟的是客户端，所以 Vuser 和真实的客户端一样，和离它最近的 Server 打交道，而和整个系统的功能无关。比如一个 Web Mail 系统，虽然它是实现了 Mail 的功能，但作为 Client 端的 IE 直接连接的是 Web Server 而不是 Mail Server，在这种情况下我们只考虑 IE 和 Web Server 的通信方式，也就是 HTTP 协议。

选择第二定律：匹配原则由高协议到低协议。

确定了被测系统的网络协议后，选择 LoadRunner Vuser 应该以实际网络协议

匹配为佳，如不能匹配，LoadRunner Vuser 则降一个协议层次再次匹配。

比如，我们如果碰到一个网络应用，其采用的应用协议比较独特，在 LoadRunner 里没有找到合适的协议，那么我们就降低一个层次，用 Winsock 来录制，那是肯定没有问题的。因为几乎所有的网络传输中都是基于 TCP 协议或 UDP 协议的，而 Socket 协议正是这一级上的协议。但是由于 Socket 协议级别太低，你录下来的东西是很难理解的，都是 Socket、Port、Data 之类的东西。所以，尽量用高层协议来录制，我们就能看懂了。

为什么要这样选择 Vuser 类型，这和 LoadRunner 的录制技术有关，下面我们就看看 LoadRunner 是如何录制的。

3.2.3 LoadRunner 录制技术

Vuser 脚本开发是 LoadRunner 做性能测试一个很关键的任务。Vuser 脚本通过执行对服务器 API 的调用来直接与服务器通信，而不需要依赖客户端软件。这样，甚至在客户端软件的用户界面完全开发好之前，便可以使用 Vuser 来检查服务器性能。

能够录制脚本是 LoadRunner 提供给用户的一项非常重要的功能，它使生成脚本的工作变得简单，用户先录制，然后在自动产生的脚本的基础上进行修改，从而快速开发出一个逻辑功能和客户端软件完全一样的压力脚本程序。这会大大减少性能测试中开发脚本的难度和工作量。

我们在感受录制脚本方便的同时，也会觉得 Vuser 很“聪明”，人工操作一遍，Vuser 就能根据我们的操作生成相应的脚本。那么 Vuser 是怎么做到的呢？

1. 录制实例

Vuser 通过录制客户端和后台服务器之间的通信包，分析其中的协议，自动产生脚本。

录制技术主要是通过 Proxy 的方式来实现的，如图 3-8 所示。

在录制时，用户在操作客户端的同时，Vugen 会捕获软件客户端到服务器的网络传输数据，然后对捕获的数据包进行拆包。由于协议的不同就是体现在数据包的结构不同上，LR 通过对包结构的分析，判断是不是它支持的协议，对包数据的分析，来获取用户发送的东西。

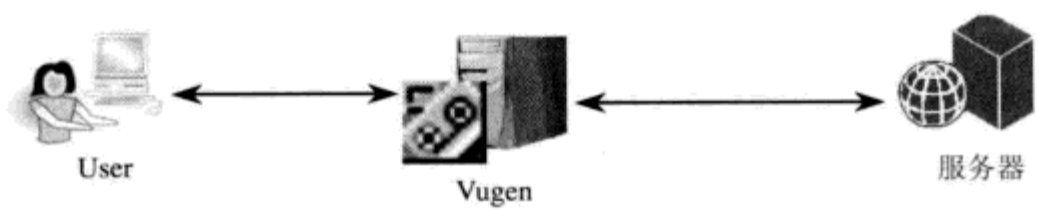


图 3-8　VU 录制原理

比如，你用 FTP 协议去录制一个访问网页的 IE 操作，那肯定是无所收获的。因为 LR 没有在网络中截获到 FTP 协议格式的包，都是 HTTP 协议格式的包，它不认，当然就是一个录制为空的结果了。

如果我们选择了 LoadRunner 的 Winsock Vuser，去录制一个 FTP 登录的操作，FTP Server 是 192.168.1.1，FTP 端口号是 21，那么 VU 在录制之后，会将登录的操作以 Socket 方式记录下来，生成 Socket 层次的脚本，类似如下示例：

```
    Action()
    {
        lrs_create_socket("socket1",      "TCP",      "LocalHost=0",
"RemoteHost=196.168.1.1:21",  LrsLastArg);
        lrs_receive("socket1", "buf2", LrsLastArg);
        lrs_send("socket1", "buf3", LrsLastArg);
        lrs_receive("socket1", "buf4", LrsLastArg);
        lrs_send("socket1", "buf5", LrsLastArg);
        lrs_receive("socket1", "buf6", LrsLastArg);
        lrs_send("socket1", "buf7", LrsLastArg);
        lrs_receive("socket1", "buf8", LrsLastArg);
        lrs_send("socket1", "buf9", LrsLastArg);
        lrs_receive("socket1", "buf10", LrsLastArg);
        lrs_close_socket("socket3");
        return 0;
    }
```

可以看到一次 FTP 登录，其实是在 Socket 层次上，按照 Socket“发-收-发-收-

发-收”完成了 10 次交互。其中 buf2、buf3、buf4 等为 Socket 数据缓存区，包含着录制时捕获的数据包。我们查看一下 buffer，可以看到如下内容：

buf2 是客户端发出建立 FTP Socket 连接请求之后，服务器返回的消息：

```
recv  buf2 148
"220-FileZilla Server version 0.9.11 beta\r\n"
"220-written by Tim Kosse (Tim.Kosse@gmx.de)\r\n"
"220 Please visit http://sourceforge.net/projects/filezilla/\r\n"
```

recv——这是一个从 Server 接收的数据，数据存放在名为“buf2”的数组变量里。

148——buf2 的长度是 148 个字节。

220——是 FTP Server 返回的状态码，它的意思是“Service ready for new user.”，这说明 FTP Server 已经准备好，等待客户端发登录请求了。其余的是 Server 的一些描述信息。

buf3 是客户端向 FTP Server 发送的登录请求：

```
send  buf3 11
"USER test\r\n"
```

可以看到，这个 buf3 的长度是 11 个字节，内容就是“USER test\r\n”，登录用户名为“test”，注意：“\r”和“\n”是转义字符，为命令换行，因此它们各自只占用一个字节。

FTP Server 返回 buf4，信息如下，要求输入密码：

```
recv  buf4 35
"331 Password required for cdctest\r\n"
```

客户端紧接着发送密码信息，如 buf5：

```
send  buf5 11
"PASS test\r\n"
```

Server 又返回信息：

```
recv  buf6 15
"230 Logged on\r\n"
```

230——说明用 Server 已经确认用户 FTP 登录成功。

buf7，buf8，buf9 是获取目录信息的请求和响应，不再一一分析。

可以看到，Socket 脚本按照 Socket 的交互过程来回放，但是它的最大缺点是处于太底层。要分析和修改 Socket API 的脚本以及数据包的具体内容将是一个繁重而且烦人的工作，进行关联的工作的难度也将大大提高（关联是一个重要的概念，我们将在后面详细叙述）。

因此，Vugen 应该尽可能地产生更高层网络协议的脚本，方便用户的阅读和修改工作。

下面我们再用 FTP 协议录制同样的 FTP 登录操作：

```
// Logon to the FTP server
ftp_logon("FTP",
          "URL=ftp://user:pwd@192.168.1.1:21",
          "LocalAddr=0",
          LAST);
```

这个 ftp_logon 函数的调用将产生 TCP 层的 10 次收发动作，由此可见，高层协议录制产生的脚本更加简单，而且更加容易被我们理解。

当然，Socket 方式是一切应用层协议的基础，Socket 脚本是一种通用的方式。对于 Vugen 不支持的应用层协议，只能通过 Socket 层次来录制。因此 LoadRunner 的 Winsock 协议可算得上是一个万能协议，但它的使用概率却可能是 LoadRunner 协议族中最低的。一方面是 LoadRunner 的协议基本覆盖到所有的主流软件系统网络架构，除非特殊的网络应用，用户一般都能在 LoadRunner 协议族中找到合适的解决方案；另一方面，Winsock 协议由于其过于底层，决定了它“什么都能干”，但其实是“什么都干不好”的命运。

2. 食客问题

所谓的关联（Correlation）就是把脚本中某些写死的（hard-coded）数据，转变成是撷取自服务器所送的、动态的、每次都不一样的数据。如果这样说还不太明白的话，可以想象，我们去北京一家小吃店吃饭，我点了一碗鸭血粉丝汤（南京著名的小吃），店里的伙计记下我点的菜，然后给我一张带有数字的小票，比如 97 号，过了一刻钟，伙计喊“97 号好了”，于是，我可以拿着这张 97 号小票去出货窗口，伙计看到 97 号小票，跟底票对照，OK，给我一碗鸭血粉丝汤。我于是乎就可以享用我的鸭血粉丝汤了。可以预见的是，吃完之后，如果我拿着 97 号小票去窗口试图再领一碗鸭血粉丝汤，不出意外的话，会遭来饭店伙计的白眼。

我们完全可以把这个食客的场景换成客户端-服务器交互场景，服务器就是一个饭店，食客就是客户端。第一次用户操作，VU 开始录制，服务器发给客户端一张带有数字的“小票”，客户端凭这张“小票”去服务器要求服务，这个过程还有“小票”上的数字都被 VU 录制下来（叫做 hard-coded），生成了脚本，一切 OK。但当脚本回放的时候，出问题了，客户端拿着一张“过期”的“小票”向服务器请求服务时，毫无疑问，服务器毫不留情地拒绝了这次服务。

显而易见，要避免出现“食客问题“，就不能 hard-code，就要保证脚本每次运行时，客户端的“小票”上的数字与服务器最新提供给它的小票数字保持一致。解决方法就是我们本节要提到的关联，即建立客户端和服务器数据的联系。其原理是第一次客户端接收服务器的数据时，将“小票”保存在一个变量中，然后客户端在后面的发向 Server 的请求中，从变量中取出“小票”发给服务器。

来，让我们通过一个实际的例子来看看关联是如何实现的。

在 Web 系统架构中，浏览器向 Web 服务器发送第一个请求后，在服务器的回应数据包中，就有一个“小票”，不过在这里不叫“小票”，而是叫做 Session ID。以后浏览器再向服务器发送请求时，都会在请求数据包中夹带这个 Session ID，Web 服务器接下来就会利用这个 Session ID 来辨识跟它要数据的是不是同一个浏

览器。对于每个新的交易，服务器都会产生新的 Session ID 给浏览器。如果 Vugen 还是用旧的 Session ID 向服务器要数据，服务器会发现这个 Session ID 是失效的或是它根本不认识这个 Session ID，当然就不会传送正确的网页数据给 Vugen 了。

下面的图 3-9 说明了这样的情形。

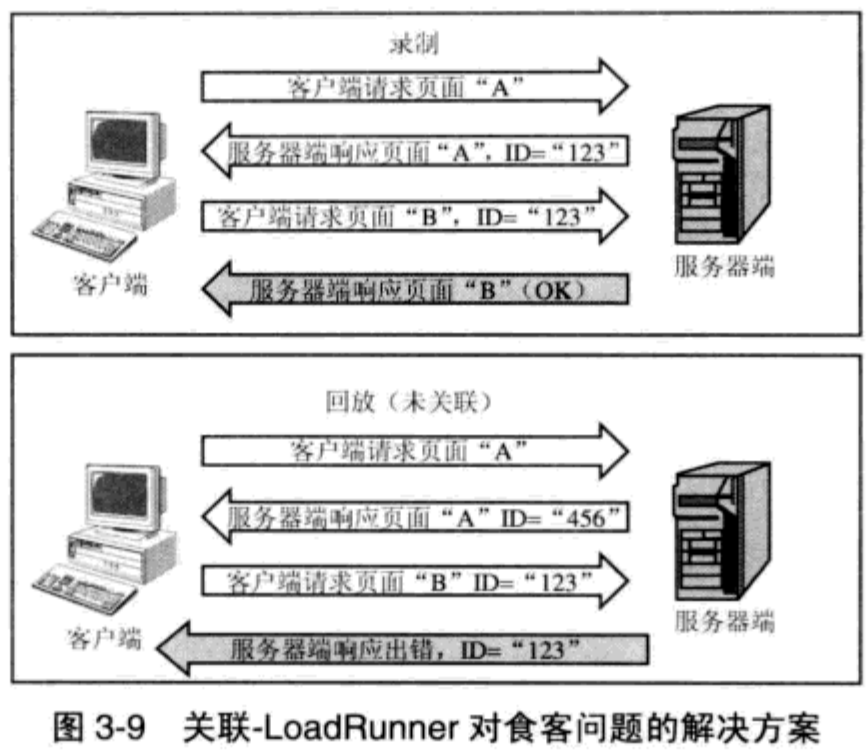


图 3-9　关联-LoadRunner 对食客问题的解决方案

要对付这种服务器，我们必须想办法找出这个 Session ID 到底是什么，位于何处，然后把它撷取下来，放到某个参数中，并且取代掉脚本中有用到 Session ID 的部分，这样就可以成功骗过服务器，正确地完成整个交易了。

关联是 Vugen 开发脚本一个非常重要的功能。因此我们在 VU 一章中将会详细地介绍如何使用关联。



第4章

LoadRunner脚本语言基础

在用 LoadRunner 做性能测试的过程中，编写脚本是一项非常重要的工作。不夸张地说，一个脚本的好坏关系到性能测试的成败。“工欲善其事，必先利其器”，在本章我们将学习 LoadRunner 脚本语言以及脚本的开发技巧。

下面我们要通过一个 HTTP 协议的脚本实例，来熟悉一下 VU 的脚本语言和脚本结构。

首先，我们还是启动 VU，在“File”菜单中选择“New”，然后在弹出的协议列表中选择 Web（HTTP/HTML）协议。

新建脚本后，会弹出一个录制选项设置对话框，在这个对话框里可以设置录制的内容和方式，我们在“URL Address”里输入 http://newtours.demoaut.com/，然后单击“OK”按钮。

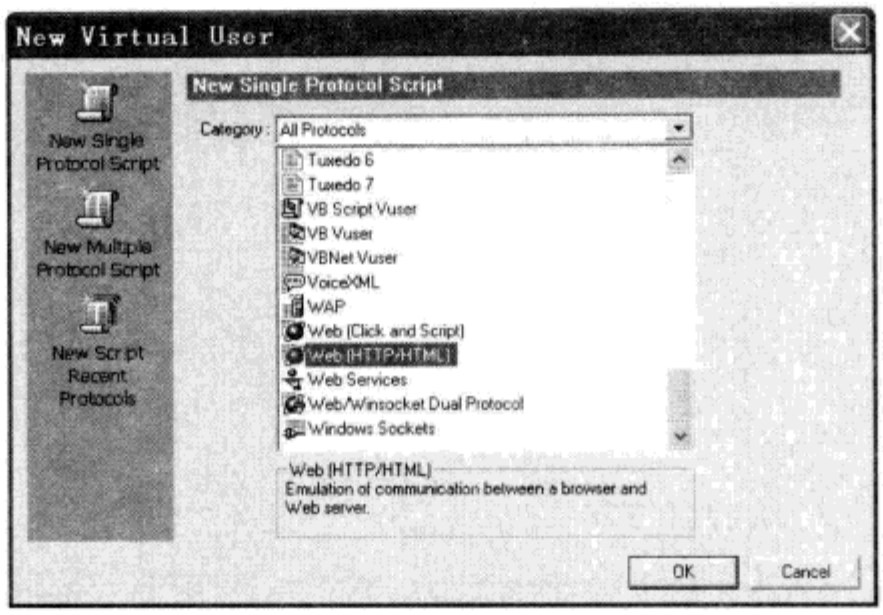


图 4-1　VU 录制之选择协议

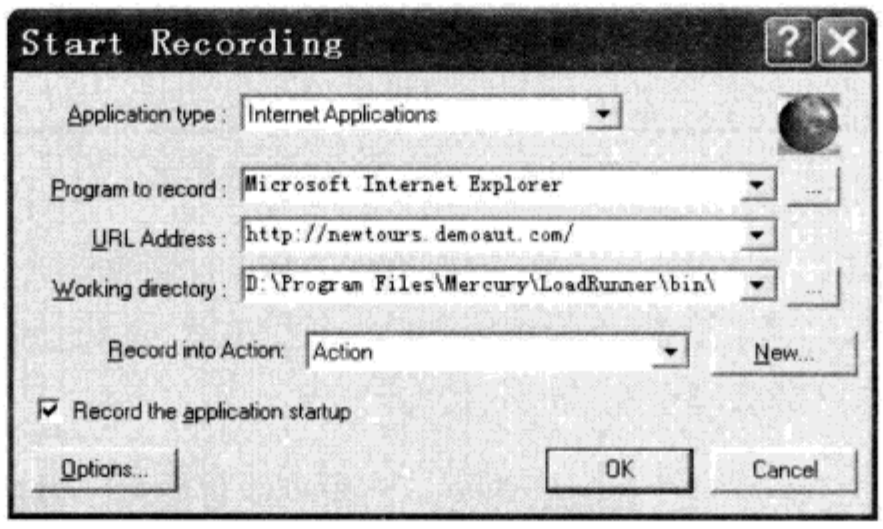


图 4-2　VU 录制之设置录制选项

“http://newtours.demoaut.com/”是 HP Mercury 提供的演示网站，在访问之前，确保你的计算机能够连接上 Internet。你也可以输入其他网址，这样 LoadRunner 就会自动捕捉浏览器访问网络的操作，记录生成一个最简单的 HTTP 脚本。

记住，在页面完全展现后，单击“Stop”按钮，停止录制，于是你就可以看到 VU 生成了如图 4-3 所示的一个脚本。

请注意，在 VU 左边导航栏中，我们看到有三个 LoadRunner 的框架函数，即 vuser_init()、Action()和 user_end()。这三个函数存在于任何 Vuser 类型的脚本中，这三个函数究竟有什么作用呢？下面我们一一分析。



图 4-3 VU 录制之生成脚本

vuser_init：顾名思义，这是虚拟用户的初始化函数，一般将用户初始化的操作放在这里，如登录操作、分配内存等，而且在做 vuser_init 的时候，Controller 的 Vuser 状态区域会显示 initialize 状态。

Action：是虚拟用户要做的业务。用户的业务操作，也就是测试内容的主体。在 VU 里设置迭代循环选项时，只针对 Action 有效，Action 会被重复运行，而 init 和 end 部分则在脚本的运行过程中只会运行一次。

vuser_end：与 vuser_init 相对应，vuser_end 做收尾工作。在 vuser_init 中如果是登录，vuser_end 里就要做退出登录；在 vuser_init 中如果是申请内存，比如使用了 malloc 函数，在 vuser_end 中就应该是释放内存，使用 free 函数。

当脚本启动时，这三个函数的运行顺序如图 4-4 所示。

实例

场景需求：我们需要对邮件系统做收发 Mail 的负载测试，每个虚拟用户登录邮件系统，收发 50 封邮件，然后退出。

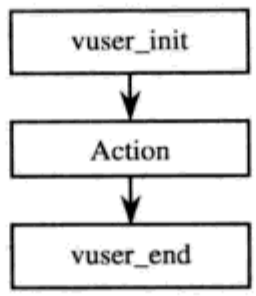


图 4-4　VU 三个函数的运行顺序

脚本设计：将登录邮件系统的操作放在 vuser_init 函数中，收发 Mail 的操作放在 Action 中，退出邮件系统的操作放在 vuser_end 中，然后将迭代循环次数设为 50。这样设计的脚本，会先运行 vuser_init 一次，然后 Actions 循环运行 50 次，再运行 vuser_end 一次，脚本结束，达到了场景需求。



在处理使用 Java 类的 Vuser 脚本时，所有的代码都将置于 Actions 类中。而 Actions 类同样包含了三个方法：init、action 和 end。

在 VU 右侧脚本编辑框中，我们看到的是 Action 的实现主体，在运行时，Action 内的 web_url 函数会被调用运行。

在 LoadRunner 的脚本中，我们可以调用三种函数：

（1）VU 通用函数，一般以 lr 开头，就如上面的 lr_start_transaction 函数。

（2）协议相关函数，不同类型的 Vuser 的函数一般以本协议类型开头。如上面的脚本是 Web（HTTP/HTML）类型的，web_url 就是一个协议函数，web 前缀说明它是属于 Web HTTP 协议的，HTTP 协议函数还包括 web_list、web_link。

（3）语言相关函数。上面的 VU 脚本是用 C 语言写的，那么 C 语言的标准函数或 dll 都可以在这里被加载和使用。

一般地，VU 录制生成的脚本中包括前两种函数，如果要对脚本进行扩充，可使用第三种“用户自定义函数”开发，这是 Java 脚本里经常做的事情，这三种函数一起构成了 VU 的 API，如图 4-5 所示。这些 API 组合起来的威力是强大的，尤

其 C 语言的扩展性和灵活性，能够保证我们开发出来不同网络协议的脚本，使得 VU 与 Server 进行交互。

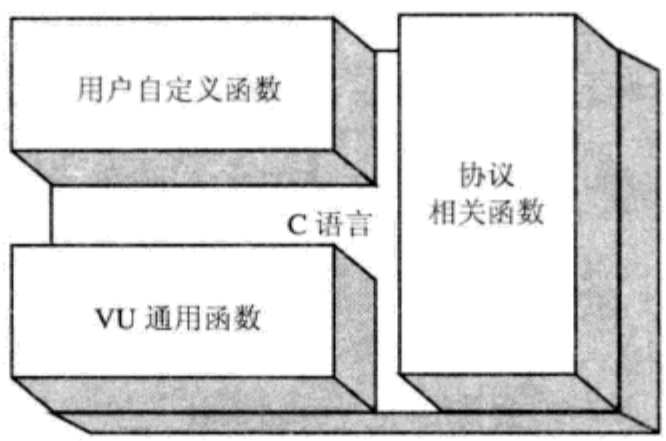


图 4-5 LoadRunner API 结构图



在 LoadRunner 早期版本中，VU 只支持 C 语言，从 8.0 开始，VU 的功能进一步增强，开始支持 Java、VB、VBScript 和 Java Script。

在这里，我们以 C 语言为例，介绍一下 LoadRunner 的 API。

4.1 C 语言与 LoadRunner 脚本

在默认情况下，Vugen 的自动脚本生成器使用 C 语言为大多数协议创建 Vuser 脚本。

在本节我们主要学习一下 LoadRunner 协议中最普遍支持和使用的 C 脚本。这是我们录制、开发 LoadRunner 脚本所必须掌握的基本功，一旦我们掌握了 C 语言的奥妙，再复杂强大的 LoadRunner 脚本也不在话下。

LoadRunner VU 支持 C 语言，所有标准 ANSI-C 约定都适用于 C Vuser 脚本，包括控制流和语法。与在其他 C 程序中的操作类似，可以向脚本中添加注释和条件语句。可以使用 ANSI-C 约定声明和定义变量。

C 语言是一种结构化语言。它层次清晰，便于按模块化方式组织程序，易于

调试和维护。C 语言的表现能力和处理能力极强。它不仅具有丰富的运算符和数据类型，便于实现各类复杂的数据结构；它还可以直接访问内存的物理地址，进行位（bit）一级的操作。这种灵活性和高效率的优势同样也体现在 LoadRunner 的 C 语言脚本里。

在这里，我们先介绍一下 C 语言的基本知识。

4.1.1　看不见的 main

在 C 语言里，main 是主函数的函数名，表示这是一个主函数。每一个 C 源程序都必须有，且只能有一个主函数（main 函数）。程序的执行总是从 main 函数开始，完成对其他函数的调用后再返回到 main。比如下面一个简单的 C 程序：

```
#include<stdio.h>
void my_action()
{
/* 在屏幕输出伟大的 hello world */
printf("hello world");
}
int main()
{
    my_action();
    return 0;
}
```

include 是预处理命令，其作用是将外部函数引用到本文件。

“/*”是 C 语言的注释开始符，一个注释块是以“/*”开头并以“*/”结尾的串。在“/*”和“*/”之间的即为注释。程序编译时，不对注释做任何处理。注释可出现在程序中的任何位置。注释用来向用户提示或解释程序的意义。在调试程序中对暂不使用的语句也可用注释符括起来，使编译跳过不做处理，待调试结束后再去掉注释符。

本程序运行时，程序首先找到 main 的入口，然后调用 my_action()函数，在 my_action()函数中向屏幕输出“hello world”文字，my_action()执行完后，回到 main，退出。程序执行结果如下，在屏幕上输出：

```
hello world
```

printf 的功能是向屏幕输出变量内容。在 LoadRunner 的 API 里也有一个和 printf 功能一样的函数：lr_output_message，我们如果在 VU 的 vuser_init、Action 和 vuser_end 中各加一条 lr_output_message 语句来标明自己所在，如下所示。

vuser_init 做如下修改：

```
vuser_init()
{
        lr_output_message("I am in vuser_init!");
        return 0;
}
Action()
{
        lr_output_message("I am in action!");
        return 0;
}
vuser_end()
{
        lr_output_message("I am in vuser_end!");
        return 0;
}
```

修改完毕后，单击“运行”按扭，开始运行脚本，在输出窗口中会看到如下结果：

```
Virtual User Script started
Starting action vuser_init.
Web Turbo Replay of LoadRunner 8.1.0 for WINXP; WebReplay8 build
5187  [MsgId: MMSG-27143]
Run-Time Settings file: "C:\example\default.cfg" [MsgId: MMSG-27141]
vuser_init.c(3): I am in vuser_init!
Ending action vuser_init.
Running Vuser...
Starting iteration 1.
Starting action Action.
Action.c(3): I am in action!
Ending action Action.
Ending iteration 1.
```

```
Ending Vuser...
Starting action vuser_end.
vuser_end.c(3): I am in vuser_end!
Ending action vuser_end.
Vuser Terminated.
```

我们发现运行结果除了输出“I am in vuser_init”、“I am in Action”和“I am in vuser_end”三条语句之外，还有其他一些输出语句不是我们控制输出的。它们其实是 Vuser 的框架 log，无论运行什么脚本，它们总会出现在输出窗口里，以提示当前的运行状态。

Iteration 是我们在 LoadRunner 运行时设置的 Action 迭代次数，默认为 1。从以上输出结果可以看出来，LoadRunner 的运行机制是先运行 vuser_init，然后根据设置的迭代次数运行 Action，最后调用 vuser_end，退出。

但在 LoadRunner 脚本里，我们没有看到 C 程序的入口 main 函数，难道 LoadRunner 脚本不需要 main 函数么？

其实 C 程序是不可能没有 main 的，只不过这个 main 函数和 LoadRunner 的实现细节一起已经被隐藏在后台，负责幕后操作，而用户可见的是 LoadRunner 给用户已经封装好的函数和接口。

可以想象，LoadRunner 的实现是这样的：

```
/*用户可见部分*/
int vuser_init()
{
……//用户填写初始化
}
int action();
{
……//用户填写
}
int vuser_end()
{
 ……//用户填写
}
/*后台隐藏部分*/
int main()
{
```

```
/* 初始化脚本全局变量和环境，调用 vuser_init*/
vuser_init();

/*调用 Action*/
Action();

/*调用 vuser_end*/
vuser_end();
return 0;
}
```



提示 vuser_init、Action 和 vuser_end 其实也是三个普通的函数，只是 LoadRunner 赋予了它们功能的含义。我们如果运行下面这段脚本，可能会看得更清楚一些。

```
vuser_init()
{
//清除浏览器缓存
//登录
.....
}
Action()
{
//trade 事务开始
Lr_start_transaction("trade");
    //web_reg_find 是预先注册一个查找，在后面的请求返回的结果内查找指定的字符串，其中查找的左边界是 LB，右边界是 RB。如果出现了 welcome 的字样，则设交易为成功，否则为失败
    web_reg_find("TextCheckPoint",
    "NOTFOUND=ERROR",
    "LB=Welcome to ",
    "RB= Manager",
    LAST);
 ......................
 ......................

lr_output_message("TextCheckPoint:%s", lr_eval_string("{TextCheckPoint}"));
    //如果我们查找成功，那么脚本继续，如果查找失败，那么将会调用 vuser_init();
```

```
If (strcmp (lr_eval_string ("{TextCheckPoint}"),"Debt")==0)
{
    lr_log_message("*******TextCheckPoint OK********");
}
else
{
    lr_end_transaction("trade ", LR_FAIL);
    lr_log_message("********TextCheckPoint FAIL***********");
        //失败后，重新运行 vuser_init，清除缓存
        vuser_init();
}
  lr_end_transaction("trade", LR_AUTO);
}
```

在上面的脚本中，vuser_init()函数作为一个普通的函数，在 Action 函数中被调用。web_reg_find 和 lr_eval_string 这两个函数是 LoadRunner 脚本常用来存储和读取参数的两个函数，它们由 LoadRunner 提供，我们将在后面的章节陆续为大家介绍。

4.1.2　全局变量与局部变量

C 语言中的变量，按作用范围可分为全局变量和局部变量。局部变量也称为内部变量。局部变量是在函数内作定义说明的，其作用域仅限于函数内，离开该函数后再使用这种变量是非法的。例如：

```
    int m,n;
    //m,n 定义在函数体外，为全局变量，作用域为整个文件，在 main,f1,f2 函数中均可被使用
    int f1(int a) /*函数 f1*/
    {
        int b,c;
    }
        //a,b,c 为局部变量，作用域仅在 f1 函数体内，在 f1 函数之外再使用 a,b,c 变量就会报错

    int f2(int x) /*函数 f2*/
    {
    int y,z;
    }
```

```
    //x,y,z 为局部变量，作用域仅在 f2 函数体内，在 f2 函数之外再使用 x,y,z
变量就会报错
main()
{
    m=4;
    n=5;
    f1(m);
    f2(n);
    }
```

在 LoadRunner 中，vuser_init、Action、vuser_end 和上面例子中的 f1、f2 函数一样，也都是普通的函数。那么在 vuser_init 中定义的变量是局部变量，其作用域在 vuser_init 函数范围内，在 vuser_init 函数外试图使用它就会发生错误。

假设我们有这样一个需求：用户在初始化的时候需要分配内存，在用户退出的时候释放内存，那么下面的代码会出现错误：

```
vuser_init()
{
    char *p = (char *)malloc(1000*sizeof(char));
}
Action()
{
}
vuser_end()
{
    free(p);
}
```

这段代码是无法编译通过的，在 vuser_init 中定义了指针 p，并为其用 malloc 函数申请了 100 个字节的内存空间，为了避免内存泄漏，用户试图在 vuser_end 中去释放 p 指向的内存空间，但 free(p)的时候，由于变量 p 是在 vuser_init 中定义的，其作用域仅限于 vuser_init 范围，离开了 vuser_init，在 vuser_end 函数中使用变量 p，就会报出“p 变量未被定义”的错误。

那如何使 vuser_init、Action 和 vuser_end 能共同使用一个变量呢？这就需要全局变量了。

全局变量也称为外部变量，它是在函数外部定义的变量。它不属于哪一个函数，它属于一个源程序文件，其作用域是整个源程序。在函数中使用全局变量，一般应作全局变量说明。

在使用 LoadRunner 的 HTTP 协议录制 Web 系统时，会生成一个 globals.h 文件，在这里定义的变量，相当于一个 LoadRunner 脚本的全局变量，可以在 vuser_init、Action、vuser_end 中被使用。

在 globals.h 文件中添加变量定义：

```
#ifndef _GLOBALS_H
#define _GLOBALS_H
//包含头文件
#include "lrun.h"
#include "web_api.h"
#include "lrw_custom_body.h"
//定义全局变量
char *p;
#endif //_GLOBALS_H
```

在 vuser_init 和 vuser_end 中增加如下代码：

```
vuser_init()
{
    p = (char *)malloc(10*sizeof(char)); //为 p 分配 10 个字节的内存空间
    memset(p,'0',9); //将前 9 个字节的空间全部赋值为"0"
    memset(p+9,'\0',1); //将第 10 个字节赋值成字符串的结束符"\0"
}
Action()
{
    lr_output_message("%s",p); //在 Action 中输出 p 指向的空间的值
    }
vuser_end()
{
    lr_output_message("%s",p);//在 vuser_end 中输出 p 指向的空间的值
   free(p); //释放内存，与 malloc 成对使用
}
```

这次脚本就能顺利编译通过和运行了。在 globals.h 中定义了一个 char 类型的

指针 p，然后在 vuser_init 中使用 malloc 函数为 p 在堆中分配了 10 个字节的空间，并将此空间全部赋值为“0”。因为 p 是全局变量，在 vuser_init、Action、vuser_end 中都可使用，因此，在 vuser_end 中用 free 函数释放 p 指向的内存空间，达到了我们的目的。

执行脚本，输出结果如下：

```
    Virtual User Script started
    Starting action vuser_init.
    Web Turbo Replay of LoadRunner 8.1.0 for WINXP; WebReplay8 build
5187  [MsgId: MMSG-27143]
    Run-Time Settings file: "C:\example\default.cfg"  [MsgId:
MMSG-27141]
    Ending action vuser_init.
    Running Vuser...
    Starting iteration 1.
    Starting action Action.
    Action.c(3): 000000000
    Ending action Action.
    Ending iteration 1.
    Ending Vuser...
    Starting action vuser_end.
    vuser_end.c(3): 000000000
    Ending action vuser_end.
    Vuser Terminated.
```

我们可以看到 vuser_init 中的字符串指针在 Action 中也照样可以被使用，输出值为 000000000，然后在 vuser_end 中 p 指针指向的内存被释放。

4.1.3 在 LoadRunner 脚本里灵活使用 C 语言

LoadRunner 脚本支持 C 语言，因此标准 C 语言的规范都可在 LoadRunner 里得到应用，包括 C 支持的数据类型、控制语句、文件处理等。

以下是一个简单的例子。

假设我们要创建一个 Web 脚本来测试电子购物站点。此站点上目前有 10 种

商品，每种商品都具有均等的可能性被客户购买。另外，网站支持 30 个用户在线购买。也就是说，我们要模拟 30 个用户登录电子购物网站，然后购买同等数量的商品。

针对这种场景，我们可以想到的有两种解决方法：第一种方法，录制 10 个脚本，每个脚本完成 1 种网上商品的订购，相应地，设置 10 个虚拟用户组（Vuser group），每组三个虚拟用户（Vuser），这样保证一共 30 个用户，每种商品的购买数量相同。第二种方法，我们创建一个脚本，在脚本内部用程序实现分派同等数量的用户去购买不同的商品。显然如果第二种能够实现，将会降低测试的复杂度，在这种模式下，启用一个脚本、一个虚拟用户组即可完成工作。

下面，我们尝试使用第二种方法来实现脚本。

在实现脚本之前，我们需要了解以下几个 C 语言的知识点：

（1）指针

指针是C语言中广泛使用的一种数据类型。运用指针编程是C语言最主要的风格之一。利用指针变量可以表示各种数据结构；能很方便地使用数组和字符串；并能像汇编语言一样处理内存地址，从而编出精练而高效的程序。指针极大地丰富了C语言的功能。学习指针是学习C语言中最重要的一环，能否正确理解和使用指针是我们是否掌握C语言的一个标志。

在计算机中，所有的数据都是存放在存储器中的。一般把存储器中的一个字节称为一个内存单元，不同的数据类型所占用的内存单元数不等，如整型量占 2 个单元，字符量占 1 个单元等。为了正确地访问这些内存单元，必须为每个内存单元编上号。根据一个内存单元的编号即可准确地找到该内存单元。内存单元的编号也叫做地址。既然根据内存单元的编号或地址就可以找到所需的内存单元，所以通常也把这个地址称为指针。

指针是 C 语言灵活高效的精髓，它是一个特殊的变量，它里面存储的数值被解释成内存里的一个地址。在函数调用的时候，指针作为形参，可以作为返回值。

（2）%运算符

%运算符是返回左值和右值相余的结果。比如 10%3 的结果为 1，3%4 的结果为 3。

（3）switch/case 控制结构

switch/case 控制结构在 C 语言中用来控制程序多支流程，其对 switch 后的一个表达式的值进行判断，根据值的不同，选择不同的 case 分支。

```
switch（表达式）{
case 常量表达式 1：语句 1;
case 常量表达式 2：语句 2;
…
case 常量表达式 n：语句 n;
default：语句 n+1;
}
```

其语义是：计算表达式的值，并逐个与其后的常量表达式值相比较，当表达式的值与某个常量表达式的值相等时，即执行其后的语句，然后不再进行判断，继续执行后面所有 case 后的语句。如果表达式的值与所有 case 后的常量表达式均值不相同时，则执行 default 后的语句。

（4）lr_whoami 函数

它是 LoadRunner 提供的函数，用来得到运行时的虚拟用户的信息，包括虚拟用户 ID、group 值和场景 ID。

我们在 Action 中实现的脚本代码如下：

```
    int Vuser,ProductNumber;
    lr_whoami(&Vuser, NULL, NULL);//指针作为返回值传入形参。30 个 Vuser 并发运行此脚本时，每个 Vuser 会得到一个不同的 VuserID，其取值为 1~30 之间
    ProductNumber = Vuser%10 + 1;

    switch(ProductNumber)
    {
```

```
    case 1:
    /* 在实际运行的时候，每个 case 分支去购买不同的产品，这里我们用一条输出语句，
表明我们到达了这里*/
    lr_vuser_status_message("I am buying product1");
    break;
    case 2:
    lr_vuser_status_message("I am buying product2");
    break;
    case 3:
    lr_vuser_status_message("I am buying product3");
    break;
    case 4:
    lr_vuser_status_message("I am buying product4");
    break;
    case 5:
    lr_vuser_status_message("I am buying product5");
    break;
    case 6:
    lr_vuser_status_message("I am buying product6");
    break;
    case 7:
    lr_vuser_status_message("I am buying product7");
    break;
    case 8:
    lr_vuser_status_message("I am buying product8");
    break;
    case 9:
    lr_vuser_status_message("I am buying product9");
    break;
    case 10:
    lr_vuser_status_message("I am buying product10");
    break;
    default:
    lr_vuser_status_message("invalid product %d",ProductNumber);
    }
```

这样，通过 C 语言的控制逻辑，上面的脚本大大增强了灵活性，实现了 Vuser 购买不同产品的分派工作。



default 语句一定不能丢失，否则程序在判断完所有 case 分支还没有匹配的话，就不知道该怎么办了。



以上脚本代码需要在 Controller 中多用户并发运行才能输出正确的结果，因为 lr_whoami 在 VU 中运行返回的 Vuser ID 总是 NULL。

4.1.4 高级——用户自定义函数

用户可以把一些通用的功能定义成自己的函数，在脚本里调用。这对脚本库的维护和增强很有用处。

可以通过以下几种方式来实现自定义函数。

1. 直接引用

在 C 语言里，函数会增强程序的可读性和复用性，我们通常这样定义和使用函数：

```
float average_function(int a, int b, int c)
{
    // 计算三个整型数的平均数，并将值返回
    float AverageValue=0.0;
    AverageValue = (a+b+c)/3;
    return(AverageValue);
}
int main()
{
    float X;
    X = average_function(10,20,30);
    Printf(">>>>>>  The average is %.2f",X);
    X = average_function(20,30,40);
    printf(">>>>>>  The average is %.2f",X);
    return 0;
}
```

上面的程序在 main 函数里调用了另外一个 average_function 函数，

average_function 函数用于计算三个整数的平均值，存储在一个浮点类型的变量中，然后返回。average_function 在 main 函数之前被声明和定义，因此 main 能够获知它的存在，并去调用它。

同样，LoadRunner 里用户也可以自定义函数，如果上面的例子把 main 改成 Action 也是可以的，如下：

```
float average_function(int a, int b, int c)
{
    //计算三个整型数的平均数，并将值返回
    float AverageValue=0.0;
    AverageValue = (a+b+c)/3;
    return(AverageValue);
}
 Action()
{
    float X;
    X = average_function(10,20,30);
    lr_output_message(">>>>>>  The average is %.2f",X);
    X = average_function(20,30,40);
    lr_output_message(">>>>>>  The average is %.2f",X);
    return 0;
}
```

读者可以将此段代码直接拷贝到 Action 中运行。

2. 本地加载模式

用于运行 Vuser 脚本的 C 解释器使用“惰性”链接模式，之所以称之“惰性”，是因为不需要在从脚本运行开始就定义函数，我们只要在使用函数前对其进行定义或加载即可。例如：

```
Lr_load_dll("my.dll");
Myfunc();/*在 my.dll 中定义，可以在加载 my.dll 之后，立刻直接调用*/
```

Lr_load_dll 是 LoadRunner 提供的一个加载动态链接库的函数，能够把 DLL 加载进来，让你在脚本中直接使用 DLL 里定义好的函数，而 DLL 中的运算是编译级

的，所以效率很高。

3. include 模式

在脚本里调用多个自定义函数，可以考虑把这些函数写在一个文件里，在 vuser_init、Action 和 vuser_end 中 include 之后，就可以使用这些函数了。

4. 全局加载模式

如果我们自己开发了一个很有用的函数，比如专门从 Excel 中读取测试数据，我们想在各个类型的 Vuser 脚本中都能使用这个函数，那么现在有一种更加方便的方法：全局加载 DLL 模式。

全局加载 DLL，使其函数能够用于所有 Vuser 脚本。DLL 加载成功之后，可以调用 DLL 中定义的任何函数，而无须在脚本中对其进行声明。

要调用 DLL 中定义的函数，请执行下列操作：

向 mdrv.dat 文件（位于 LoadRunner/dat 目录，如图 4-6 所示）的相应部分中添加要加载的 DLL 列表。

例如，要在 NT 平台上为 Winsocket Vuser 加载 DLL，请向 mdrv.dat 文件中添加下列语句（黑体语句）：

```
[WinSock]
ExtPriorityType=protocol
WINNT_EXT_LIBS=wsrun32.dll
WIN95_EXT_LIBS=wsrun32.dll
LINUX_EXT_LIBS=liblrs.so
SOLARIS_EXT_LIBS=liblrs.so
HPUX_EXT_LIBS=liblrs.sl
AIX_EXT_LIBS=liblrs.so
LibCfgFunc=winsock_exten_conf
UtilityExt=lrun_api
ExtMessageQueue=0
ExtCmdLineOverwrite=-WinInet No
```

```
ExtCmdLineConc=-UsingWinInet No
WINNT_DLLS=user_dll1.dll,user_dll2.dll,…..
```

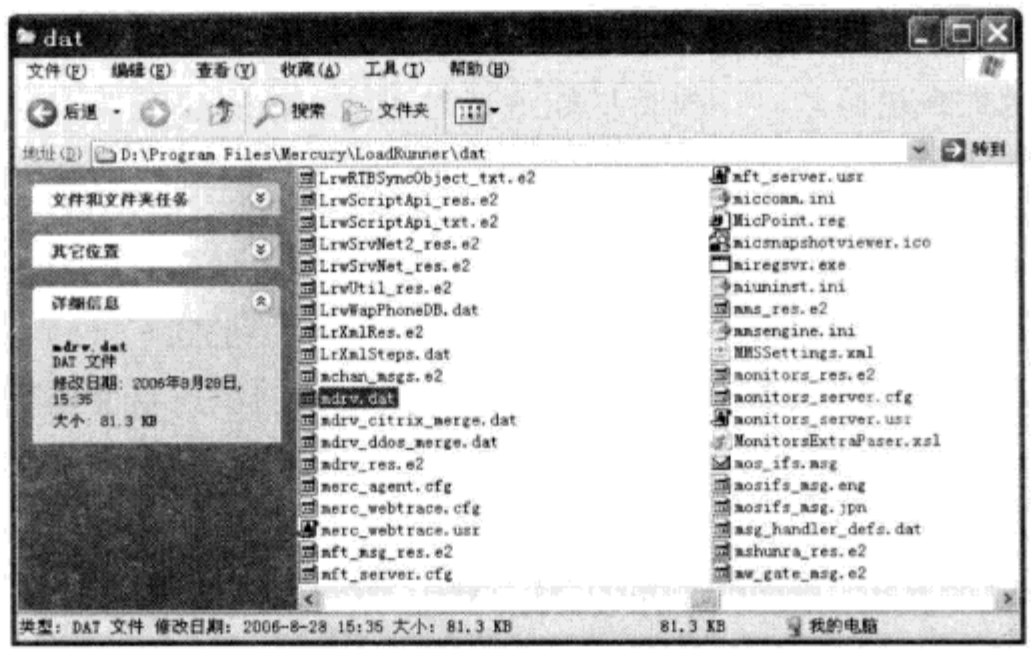


图 4-6　LoadRunner 安装目录下的 mdrv.dat 文件

然后我们在 Winsock 脚本中就可以直接使用 user_dll1 和 user_dll2 中定义的函数了。

同理，我们在 mdrv.dat 文件的其他 Vuser 配置部分中加入同样的语句，可以达到相同的目的。

4.2　通用 VU 函数

通用 VU 函数是构建在 C 语言基础之上的脚本框架函数，起支持作用。因此学习通用 VU 函数，有助于帮助我们了解 LoadRunner 脚本机制。

通用 VU 函数有一个特征，就是以 LR 开头，以标明它们属于 LR，而不专属于任何协议，只要是 C 脚本，就都可使用。

通用 VU 函数可分为事务控制函数、命令行分析函数、系统信息函数、字符串函数、日志函数和运行时函数。

4.2.1 事务和事务控制函数

软件系统的性能就是靠一个个事务来度量的。LoadRunner 中事务的定义就是靠事务函数来进行的，在执行性能测试时，LoadRunner 会采集完成其定义的每个事务所花费的时间，并在性能测试后在 Analysis 中进行统计分析。

- lr_end_sub_transaction，标记子事务的结束以便进行性能分析。
- lr_end_transaction，标记 LoadRunner 事务的结束。
- lr_end_transaction_instance，标记事务实例的结束以便进行性能分析。
- lr_fail_trans_with_error，将打开事务的状态设置为 LR_FAIL 并发送错误消息。
- lr_get_trans_instance_duration，获取事务实例的持续时间（由它的句柄指定）。
- lr_get_trans_instance_wasted_time，获取事务实例浪费的时间（由它的句柄指定）。
- lr_get_transaction_duration，获取事务的持续时间（按事务的名称）。
- lr_get_transaction_think_time，获取事务的思考时间（按事务的名称）。
- lr_get_transaction_wasted_time，获取事务浪费的时间（按事务的名称）。
- lr_resume_transaction，继续收集事务数据以便进行性能分析。
- lr_resume_transaction_instance，继续收集事务实例数据以便进行性能分析。
- lr_set_transaction_instance_status，设置事务实例的状态。
- lr_set_transaction_status，设置打开事务的状态。
- lr_set_transaction_status_by_name，设置事务的状态。
- lr_start_sub_transaction，标记子事务的开始。
- lr_start_transaction，标记事务的开始。
- lr_start_transaction_instance，启动嵌套事务（由它的父事务的句柄指定）。
- lr_stop_transaction，停止事务数据的收集。
- lr_stop_transaction_instance，停止事务（由它的句柄指定）数据的收集。
- lr_wasted_time，消除所有打开事务浪费的时间。

4.2.2　命令行分析函数

当 LoadRunner 用命令行方式启动和运行时，以下函数用来分析命令行，以得到命令行中的参数信息。

- lr_get_attrib_double，检索脚本命令行中使用的 double 类型变量。
- lr_get_attrib_long，检索脚本命令行中使用的 long 类型变量。
- lr_get_attrib_string，检索脚本命令行中使用的字符串。

4.2.3　系统信息函数

用来得到 VU 的系统信息。

- lr_user_data_point，记录用户定义的数据采集点。
- lr_whoami，将有关 Vuser 的信息返回给 Vuser 脚本。
- lr_get_host_name，返回执行 Vuser 脚本的主机名。
- lr_get_master_host_name，返回运行 LoadRunner Controller 的计算机名。

4.2.4　字符串函数

主要是对参数进行操作，包括对不同类型参数的读取、存储和移动。

- lr_eval_string，返回参数的当前值。
- lr_save_string，将以 NULL 结尾的字符串保存到参数中。
- lr_save_var，将变长字符串保存到参数中。
- lr_save_datetime，将当前日期和时间保存到参数中。
- lr_advance_param，前进到下一个可用参数。
- lr_decrypt，解密已编码的字符串。
- lr_eval_string_ext，lr_eval_string 的扩展，为指向包含参数数据的缓冲区的指针。
- lr_eval_string_ext_free，释放由 lr_eval_string_ext 分配的指针。
- lr_save_searched_string，在缓冲区中搜索字符串实例，并将该字符串实例保

存到参数中。

4.2.5 消息函数

是 VU 发送和记录 log 的函数。

- lr_debug_message，将调试消息发送到输出窗口。
- lr_error_message，将错误消息发送到输出窗口。
- lr_get_debug_message，得到当前的消息类。
- lr_log_message，将输出消息直接发送到 output.txt 文件，此文件位于 Vuser 脚本目录中。
- lr_output_message，将消息发送到输出窗口。
- lr_set_debug_message，为输出消息设置消息类。
- lr_vuser_status_message，生成格式化输出并将其打印到 Controller Vuser 状态区域。
- lr_message，将消息发送到 Vuser 日志和输出窗口。



注意 在 runtime 设置中有 log 级别的设置，设置不同的级别，会决定这些函数在运行时是否能够生效。

4.2.6 运行时（run-time）函数

运行时（run-time）多数是通过 VU 的 runtime 来设置的。有以下函数放在脚本中来实现，使 LoadRunner 的控制更加细致，对外更加灵敏。

- lr_load_dll，加载外部 DLL。
- lr_think_time，暂停脚本的执行，以模拟思考时间（实际用户在操作之间暂停以进行思考的时间）。
- lr_continue_on_error，指定脚本如何处理错误场景，是继续还是退出。
- lr_rendezvous，在 Vuser 脚本中设置集合点。

注：具体函数的使用方法参见 LoadRunner 的帮助手册。

在 Java 语言的脚本中，以上函数名字略有变化。可参照 LoadRunner 的 Java API 手册。

4.3　协议相关函数

除了通用 Vuser 函数以外，Vugen 还会在录制时生成特定于协议的函数，并将它们插入到 Vuser 脚本中。

特定于协议的函数是专门针对要录制的 Vuser 的类型而生成的。例如，Vugen 会将 LRD 函数插入到数据库脚本中，将 LRT 函数插入到 Tuxedo 脚本中，将 LRS 函数插入到 Windows 套接字脚本中。

在通用 VU 函数的基础上，不同协议类型的 Vuser 有自己一套协议相关的函数。一般地，协议相关的函数以本协议名字开头。如 HTTP 协议以 web 开头，MMS 协议以 mms 开头，Database 协议以 lrd 开头。下面我们以 HTTP 协议为例，介绍一下协议相关函数。

4.3.1　HTTP 协议原理

HTTP 协议的 Vuser 是模拟浏览器和 Web Server 的交互过程。

浏览器与 Web Server 的一个简单的交互过程如图 4-7 所示。

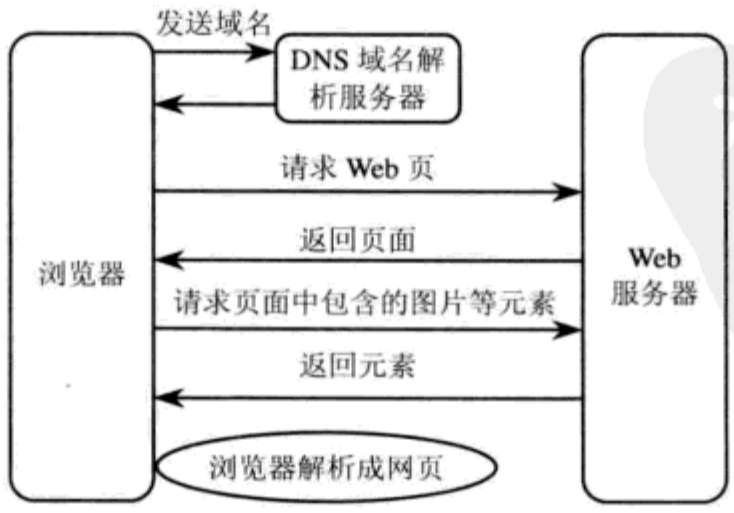


图 4-7　HTTP 协议交互原理

当浏览器想要得到一个网页时，用户在浏览器的地址栏里输入网页的 URL，然后回车，浏览器首先就会发送一个 HTTP 的命令到 Server，Server 就会把页面的数据传送给浏览器，浏览器把数据解析成我们看到的 HTML 页面。但是 HTTP 协议有一个特点，它是无状态的，也就是说，浏览器和 Server 的每个交互都是独立无关的，Server 不知道来自 Browser 的任何两个请求是否出自同一页面，或者是否有次序的规定。因此 Web 系统在 HTTP 协议上层进行了控制，通过 Cookie、Session 等机制来保证请求的状态。

4.3.2 HTTP 在 LoadRunner 的实现

以上这个过程，完全可以由 LoadRunner 的 HTTP Vuser 的函数来实现。表 4-1 列出了所要用到的 LoadRunner 函数。

表 4-1 LoadRunner HTTP 函数

内容	应用	HTTP VU 函数
HTTP 协议	Header 设置	Web_add_header Web_clean_header
	点击链接	Web_link 函数（点击文字链接） Web_image 函数（点击图片链接）
	提交表单	Web_submit_data Web_submit_form
Web 系统设置	Cache 设置	Web_cache_leanup Web_load_cache
	Cookie 设置	Web_add_cookie Web_cleanup_cookies
	Proxy 设置	Web_set_proxy Web_set_secure_proxy
LoadRunner 框架支持	检查点	Web_find Web_image_check
	关联	Web_create_html_param Web_create_html_param_ex Web_reg_save_param Web_set_max_html_param_len

续表

内容	应用	HTTP VU 函数
LoadRunner 框架支持	控制	Lr_start_transaction Lr_rendezvous Lr_think_time

由上可见，LoadRunner 提供的这套 HTTP 协议的 API 是比较细致的，不仅覆盖了浏览器与 Server 的交互，而且还有设置。

以下是一个通过 HTTP Vuser 录制生成的脚本例子：

```
#include "as_web.h"
Action1()
{
    Web_add_cookie("nav = 140, Domain = Dogbert");
    Web_url("dogbert",
          "URL = http://www.cesoo.info",
        "Reccontenttype = text/html",
        LAST);
    Web_image("library",
        "alt = library",
        LAST);
    Web_link("1 BOOK search",
        "text = 1 book search",
        LAST);
}
```

我们逐条研究，看看用户的操作是如何转换成 HTTP Vuser 函数的：

第一条语句 Web_add_cookie 的作用是保存 Server 传过来的 cookie，以后的访问都会基于此 cookie，直到脚本的结束。

第二条语句 Web_url 显示用户在 IE 地址栏中输入 www. cesoo.info，然后回车，IE 发起一个访问的请求。

用户进入 www.cesoo.info 页面后，点击了一个 alt 属性为 library 的图片，此操

作被 LoadRunner 记录为 Web_image 函数。

最后用户做了一个搜索的操作，退出。

这是一个简单的 Web 脚本，如果我们想生成功能强大一些的脚本，就需要了解 VU 的机制和原理了。

5

第5章

VU——用户行为的模拟器

LoadRunner之所以功能强大，很重要的一个因素是VU对各种网络协议的强力支持。VU作为虚拟用户的产生器，从横向上看，VU几乎支持模拟当今所有主流的软件客户端，同时还在不断地推陈出新；从纵向上看，每个Vuser脚本的设置也是非常繁多和详细的，达到了精确模拟的效果。

因此，了解并熟悉VU是我们“玩转”LoadRunner要做的第一件事情。

我们最常听到的关于VU的描述就是：**VU通过运行VU脚本模拟了用户对软件的操作行为。**

如果我们不刨根问底，探究实质，就难以发现上面这句话的奥妙。

5.1 图灵试验与 LoadRunner VU 模拟奥秘

5.1.1 图灵试验场景

伟大的计算机之父阿兰·图灵曾对人工智能设计过一个著名的“图灵试验”，来判断计算机对人类的模拟能力，图灵试验场景如图 5-1 所示。

图灵试验由计算机、被测试的人和主持试验的人组成。计算机、被测试的人和测试主持人分别在三个不同的房间内。

（1）测试过程是由主持人提出问题，由计算机和被测试人分别回答。主持人并不知道哪个答案来自于人或是计算机。

（2）被测试人回答问题是尽可能地表明他是“真正的”人。

（3）计算机也尽可能逼真地模仿人的思维方式和思维过程来回答问题。

（4）如果试验主持人听取对问题的回答后，分不清哪个是人回答的，哪个是计算机回答的，则可以认为被测试计算机是有人的智能的。

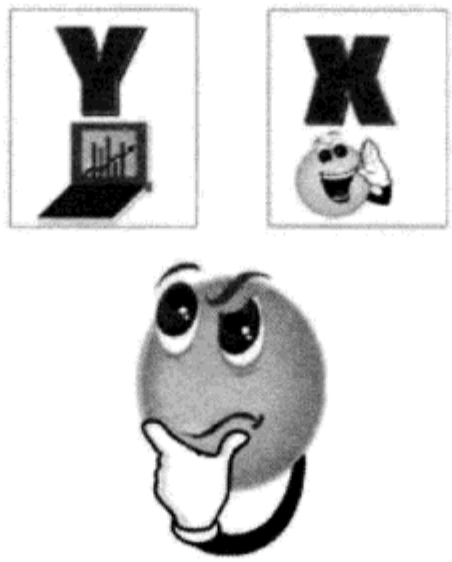


图 5-1 图灵试验场景

我们从这个试验中可以看到模拟需要有三个要素：模拟者（计算机）、被模

拟者（真实的人）和观察者（主持人）。其中观察者是很重要的一个角色，模拟只有对观察者才有意义。而在图灵试验中，观察者通过一定的方式（闻其“答案”不见其人）来判断这个模拟是否成功。

5.1.2　LoadRunner 模拟揭秘

再回头看看那句话：“VU 通过运行 VU 脚本模拟了用户对软件的操作行为。”与之对应，我们能找到 VU 是模拟者，用户是被模拟者，但这句话里没有指出观察者。那么观察者是谁？我们性能测试工程师是观察者么？显然不对，用户的操作行为是一次次键盘输入，一个个鼠标点击，而 VU 在我们眼里只是一行行脚本程序和配置文件。

想到这里，可能已经有人得到了答案。没错，观察者其实就是性能测试中被测服务器。

我们发现，完全可以把图灵试验应用到我们的 VU 模拟场景中：

在软件系统运行时，一个真实的用户通过操作软件客户端来发起和服务器的会话请求，同时，一个 VU 开始运行它的脚本，也发起了同样的请求。被测服务器在处理各个“交谈”请求并与之分别会话的时候，并不知道哪些请求是来自真实的用户，哪些是来自 VU。这样，VU 成功地模拟了真实的用户，“骗过”了被测服务器。当然，VU 要实现这个目标，之前还需要通过认证，构造合法的请求等步骤，这些 VU 都要加以实现。

我们弄明白了观察者是被测服务器这件事情后，有些知识点就清楚多了。

事实：VU 是基于网络协议的。

很明显，被测服务器是通过各种各样的网络协议与客户端打交道的。VU 要“骗过”被测服务器，当然就要遵守这些协议，按规矩、按步骤来动作，否则就会吃“闭门羹”，毫不留情地被服务器拒绝。实际上，如果我们留心的话，可以发现 VU 每个协议相关的函数都是紧紧围绕其网络协议的实现的。同样，VU 录制生成的也是网络协议层次的脚本。在回放时，VU 将按照录制下来的网

络事件，一一重放。

基于网络协议的脚本的一个好处是，我们可以使用相对少的硬件资源，来生成大量的虚拟用户负载。相比之下，WinRunner 和 QTP 的脚本是基于界面事件（GUI）的，它在一台主机上同时只能运行一个虚拟用户的脚本，因为一个虚拟用户会占用整个主机的资源。

推论一：VU 不关心用户在界面上发生的事情。

真实的用户对软件客户端可能有各种各样的操作，有些会触发对服务器的请求，有些则不会。而 VU 的录制和运行都是基于网络上的事件的，因此 VU 在录制的时候，只对那些网络事件感兴趣，如果有网络交互发生，就会记录成脚本，没有则只会生成一个空脚本，尽管可能用户在界面上做了很多操作，比如鼠标移动，填写 Web form 里的数据。

推论二：VU 中的操作关联与界面上的操作关联是不一致的

在我们对软件系统进行测试的过程中，某些操作是相关联的。例如，我们测试“查看客户交易历史明细”这个事务的系统响应时间，按界面测试流程，我们会这么做：

（1）首先用客户查询系统，查询出此客户。

（2）点击这个客户，打开另外一个页面，展现此客户的基本信息。

（3）点击基本信息中的“交易历史明细”，系统经过查询后，刷新此页面，显示明细结果。

显然，在 UI 测试中，我们要测试第三步骤，之前必须要完成第一步和第二步。但在 VU 中就不是这样了：

这三个操作被 VU 记录成三个函数。如果它们是上下文无关的函数（关于上下文相关和上下文无关函数，可参看 LoadRunner 函数手册），那么其实，我们完全可以忽略掉第一和第二个函数，只执行和参数化第三个函数。

案例分析

疑问：使用 LoadRunner 录制一个 Java applet 技术的 Web 系统，要录制的动作是：

- 点击一个按钮后，此时系统调用 applet 小应用程序。
- applet 小应用程序在本地加载完后，弹出一个通过 XML 文件配置对话框，在这个页面中建立节点树。

无论 LoadRunner 采用 Web 协议、Web Service 协议还是 Windows Socket 协议，在脚本录制并回放后，到系统中一检查，发现节点并没有创建，这是怎么回事？

解释：

根据推论一，我们知道 VU 只捕捉网络上的事件，在本案例中 LoadRunner 采用 Winsocket 协议也无法创建节点，这说明 XML 树节点的创建是由 Java applet 完成的，并且很可能只保持在客户端本地，并未有和 Server 同步的迹象。因此，LoadRunner 脚本捕捉和回放的脚本只是网络事件，并不会对客户端本地产生影响，节点自然不会被创建。

根据推论二，如果我们的目标是测试服务器的性能，那么完全可以忽略这个问题，继续录制。

提示 基于 GUI 和基于协议的测试工具在实质上是看待同一事物的不同的角度。比如：用户点击一个 Web 页面上的“无忧测试“的链接文字，在 QTP 会被记录成 link("无忧测试").click 的函数，而在 LoadRunner 中则会被记录成 web_link 函数。前者是记录“鼠标在此链接上点击一次”这个事件，而后者是“客户端向 Web Server 发起了一个 get URL 地址为 www. cesoo.info 的请求”。LoadRunner 和 QTP 如摸象的盲人一样，各自从 GUI 和网络协议的角度来看用户点击的动作。而实际上它们所记录的只是用户操作整个过程的一部分。



LoadRunner 8.1 从 SP2 开始，VU 协议族中新增了一个特殊的 Vuser 类型，叫做 Web Click and Script，为什么说它特殊呢？因为它和其他基于网络协议的 Vuser 不一样，Web Click 采用了 QTP 的技术，使得 VU 可以录制在界面上的一些操作。比如登录时填写用户名、口令这些动作都可以被记录成脚本。这在本书第 10 章中将会有所介绍。

把 VU 的实质弄清楚了之后，我们就可以放心大胆地使用它了。Vugen 是个神奇的盒子，我们总能从盒子里拿到我们想要的东西。

5.2 录制脚本

VU 通过录制用户在客户端软件的操作来直接生成脚本，用户的每个协议级的操作以 LoadRunner 的 API 函数方式记录在脚本里。回放脚本的时候，通过执行 API 函数来模拟最初用户的操作动作。

5.2.1 选择协议

选择协议的两个基本原则已经在前文介绍过了。我们这里看看 LoadRunner 具体有哪些协议。

Vuser 类型可根据应用领域分为下列几种：

- 应用程序部署解决方案：Citrix ICA。
- 客户端-服务器：DB2 CLI、DNS、Informix、MS SQL Server、ODBC、Oracle（2 层）、Sybase Ctlib、Sybase Dblib 和 Windows Sockets 协议。
- 自定义：C 模板、Visual Basic 模板、Java 模板、JavaScript 和 VBScript 类型的脚本。
- 分布式组件：适用于 COM/DCOM、Corba-Java 和 Rmi-Java 协议。
- 电子商务：FTP、LDAP、Palm、PeopleSoft 8 mulit-lingual、SOAP、Web（HTTP/HTML）和双 Web/WinSocket 协议。
- Enterprise Java Bean：EJB 测试和 Rmi-Java 协议。

- ERP/CRM：Baan、Oracle NCA、PeopleSoft-Tuxedo、SAP-Web、SAPGUI、
- Siebel-DB2 CLI、Siebel-MSSQL、Siebel-Web 和 Siebel-Oracle 协议。
- 传统：终端仿真（RTE）。
- 邮件服务：Internet 邮件访问协议（IMAP）、MS Exchange（MAPI）、POP3 和 SMTP。
- 中间件：Jacada 和 Tuxedo（6、7）协议。
- 流数据：Media Player（MMS）和 Real 协议。
- 无线：i-Mode、VoiceXML 和 WAP 协议。

当我们试图创建一个新的 Vuser 的时候，就会弹出协议选择对话框，如图 5-2 所示。

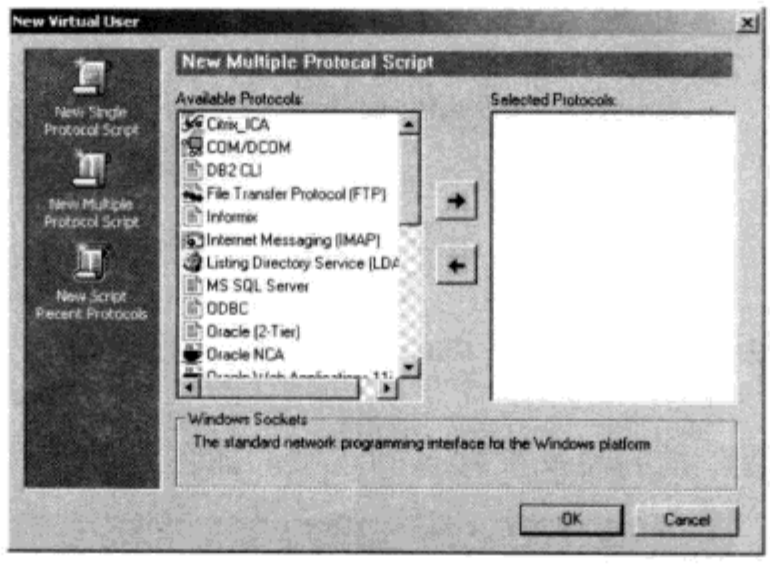


图 5-2　VU 协议选择对话框

有两种协议选择方式：单协议模式（New Single Protocol Script）和多协议模式（New Multiple Protocol Script）。

（1）单协议模式

当用户以单协议录制时，VU 只录制在既定协议上的用户操作，在 VU 中我们可以使用单协议模式选择任何一种协议。

（2）多协议模式

当用户以多协议录制时，VU 录制几个协议上的操作。并不是任意的协议都可

组合成多协议模式。有以下协议支持多协议录制：COM、FTP、IMAP、Oracle NCA、POP3、Real Player、Windows Socket（raw）、SMTP 和 Web。双协议 Web/Win Socket 应该被看做是单协议，因为其机制与多协议还是不一样的。

不同类型的 Vuser 的另外一个区别是多 Action 的支持，一些协议支持多 Action，目前这些协议是：Oracle NCA、Web、RTE、general C、WAP、I-Mode 和 voice XML。关于多 Action 是何物，又如何使用，可以参看本章 5.9 节“高级——多 Action”。

5.2.2 规划脚本结构

在录制时，用户可以选择哪些操作生成脚本在 vuser_init、Action 和 vuser_end 中，同时，也可以在录制时随时加入 transaction 的定义、注释和同步点。VU 录制工具条如图 5-3 所示。



图 5-3 VU 录制工具条

5.2.3 HTTP Vuser 中的 URL mode 和 HTML mode

在录制之前，我们需要设置录制选项，如图 5-4 所示。

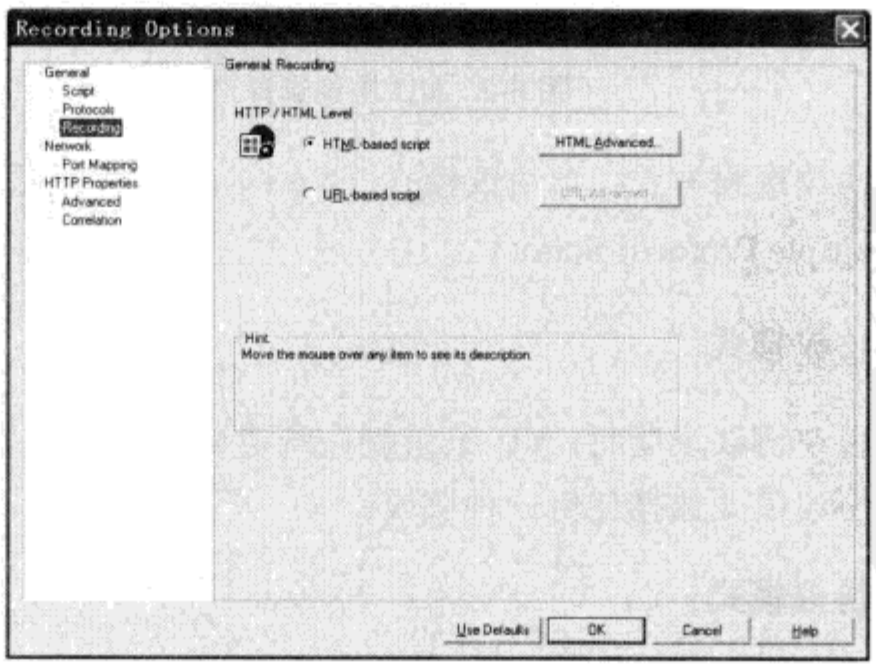


图 5-4 VU 录制设置选项

在默认情况下，选择“HTML-based script”，说明脚本中采用 HTML 页面的形式来表示，这种方式的 Script 脚本容易维护，容易理解，推荐以这种方式录制。

“URL-based script”说明脚本中的表示采用基于 URL 的方式，所有的 HTTP 的请求都会被录制下来，单独生成函数，所以 URL 模式生成的脚本会显得有些杂乱。

实例

以 HTML 模式录制，访问“http://newtours.demoaut.com/”网站会生成下面的脚本：

```
Action()
{
  web_url("newtours.demoaut.com",
    "URL=http://newtours.demoaut.com/",
    "Resource=0",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t1.inf",
    "Mode=HTML",
    LAST);
  return 0;
}
```

以 URL 模式录制同样的操作，会生成如下脚本：

```
Action()
{
  web_url("newtours.demoaut.com",
    "URL=http://newtours.demoaut.com/",
    "Resource=0",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t1.inf",
    "Mode=HTTP",
    LAST);

  web_url("logo.gif",
    "URL=http://newtours.demoaut.com/images/nav/logo.gif",
    "Resource=1",
```

```
    "RecContentType=image/gif",
    "Referer=http://newtours.demoaut.com/",
    "Snapshot=t2.inf",
    LAST);

  web_url("html.gif",
    "URL=http://newtours.demoaut.com/images/nav/html.gif",
    "Resource=1",
    "RecContentType=image/gif",
    "Referer=http://newtours.demoaut.com/",
    "Snapshot=t3.inf",
    LAST);

   web_url("boxad1.gif",
    "URL=http://newtours.demoaut.com/images/nav/boxad1.gif",
    "Resource=1",
    "RecContentType=image/gif",
    "Referer=http://newtours.demoaut.com/",
    "Snapshot=t4.inf",
    LAST);
.....................................
.....................................
//经统计，录制生成的 web_url 函数有 20 个
return 0;
}
```

是选择 HTML 还是 URL 录制，有以下参考原则：

（1）基于浏览器的应用程序推荐使用 HTML-based script。

（2）不是基于浏览器的应用程序推荐使用 URL-based script。

（3）如果基于浏览器的应用程序中包含了 JavaScript 并且该脚本向服务器产生了请求，比如 DataGrid 的分页按钮等，也要使用 URL-based script 方式录制。

（4）基于浏览器的应用程序中使用了 HTTPS 安全协议，使用 URL-based script 方式录制。

5.2.4　查看日志

在录制和回放的时候，VU 会分别把发生的事件记录成日志文件，这些日志有利于我们跟踪 VU 和服务器的交互过程。我们可以通过 VU 输出窗口观察日志，也可以到脚本目录中直接查看文件。其中有三个主要的日志对我们的录制很有用：

1．执行日志（Execution Log）

脚本运行时的输出都记在这个 Log 里。

“输出”窗口的“执行日志”显示的消息用于描述 Vuser 运行时执行的操作。该信息可说明在方案中执行脚本时，该脚本的运行方式。

脚本执行完成后，可以检查“执行日志”中的消息，以查看脚本在运行时是否发生错误。

“执行日志”中使用了不同颜色的文本。

- 黑色：标准输出消息。
- 红色：标准错误消息。
- 绿色：用引号括起来的文字字符串（例如 URL）。
- 蓝色：事务信息（开始、结束、状态和持续时间）。

如果双击以操作名开始的行，光标将会跳到生成的脚本中的相应步骤上。

图 5-5 显示了 Web Vuser 脚本运行时的“执行日志”消息。

执行日志是我们调试脚本时最有用的信息。有关设置执行日志级别调试脚本的技巧，在本章 5.11 节“高级——脚本调试技巧”中有详细介绍。

2．录制日志（Recording Log）

当录制脚本时，Vugen 会拦截 Client 端（浏览器）与 Server 端（服务器）之间的对话，并且通通记录下来，产生脚本。在 Vugen 的 Recording Log 中，我们可以找到浏览器与服务器之间所有的对话，包含通信内容、日期、时间、浏览器的

请求、服务器的响应内容等，如图 5-6 所示。脚本和 Recording Log 最大的差别在于，脚本只记录了 Client 端要对 Server 端所说的话，而 Recording Log 则是完整记录二者的对话。因此通过录制日志，我们能够更加清楚地看到客户端与服务器的交互，这对我们开发和调试脚本非常有帮助。

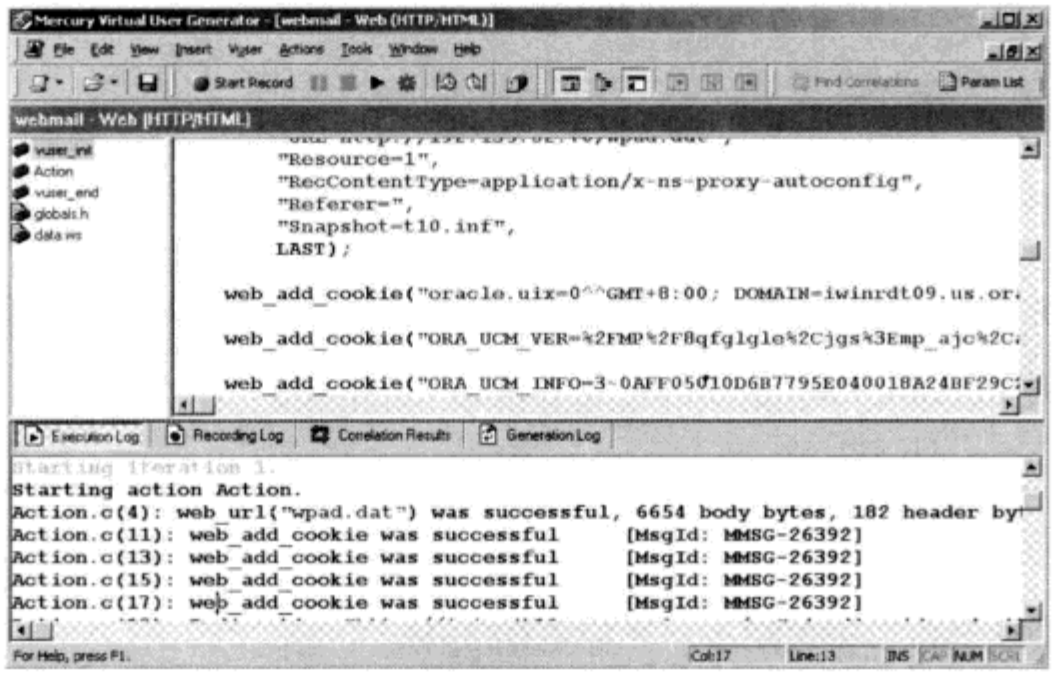


图 5-5 VU 脚本执行日志

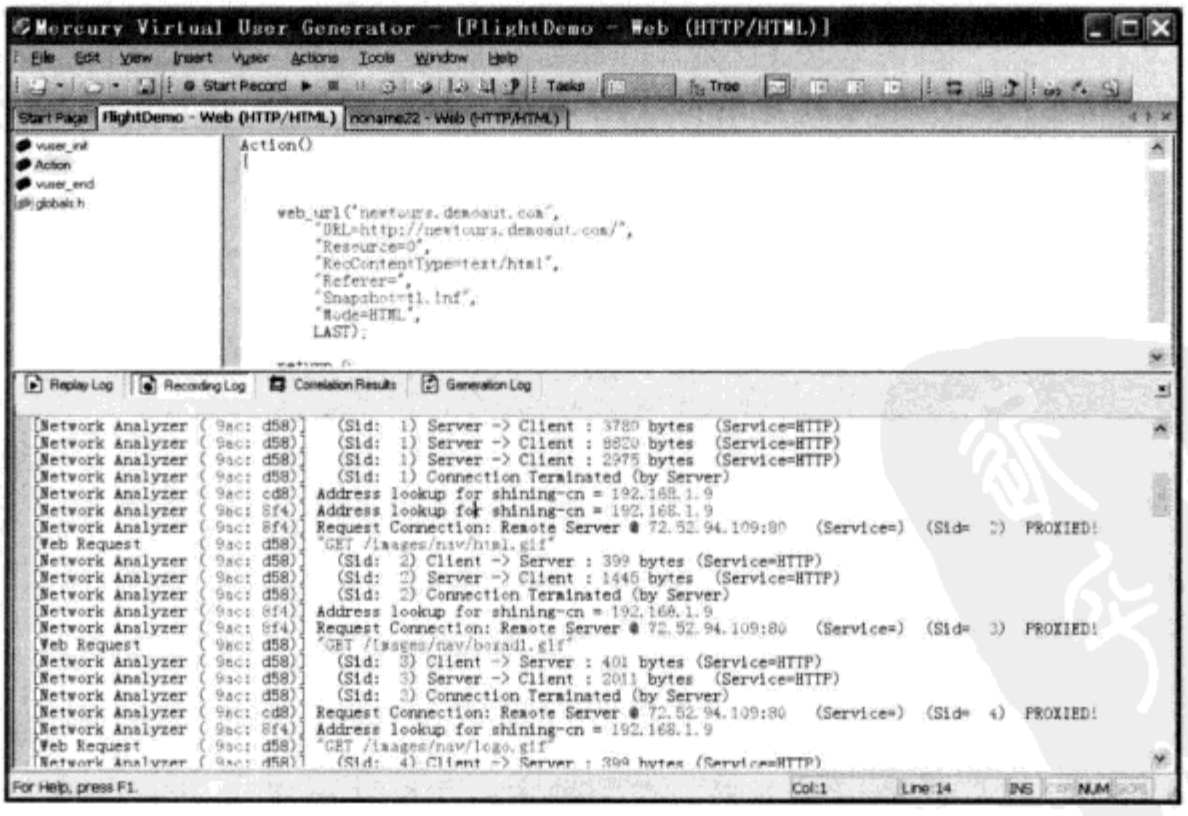


图 5-6 VU 脚本录制日志

3. 产生日志（Generation Log）

产生日志记录了脚本录制的设置、网络事件到脚本函数的转化过程。



这里同样需要注意的是：脚本能正常运行后应禁用日志。因为产生及写入日志需占用一定资源。

5.3　回放脚本

单击 run 按钮，或按快捷键“F5”就可以运行脚本。VU 脚本运行工具条如图 5-7 所示。

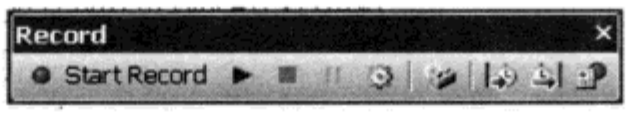


图 5-7　VU 脚本运行工具条



在 LoadRunner 8.0 中提供了编译功能，快捷键为“Shift+F5”，这样脚本本身的语法检查等简单工作可以在编译中来做，省却了每次都要运行脚本的麻烦。



VU 提供运行时查看浏览器活动的功能。这对于我们调试脚本是非常有帮助的，可以方便我们直观地查看运行状态。这有些类似 QTP 和 WinRunner 的功能，需要修改设置来开启这个功能。

设置方法如下：在 VU 菜单“Tools”>“General Options”>“Display”选项卡中，勾选“Show browser during replay”项，如图 5-8 所示。

需要注意的是，运行时开启的 Run-Time Viewer 是 LoadRunner 自带的 HTML 解释器，并不是完全的 IE，因此有些 Java Script 和 Applet 可能会不能正常显示。VU 运行时查看器如图 5-9 所示。

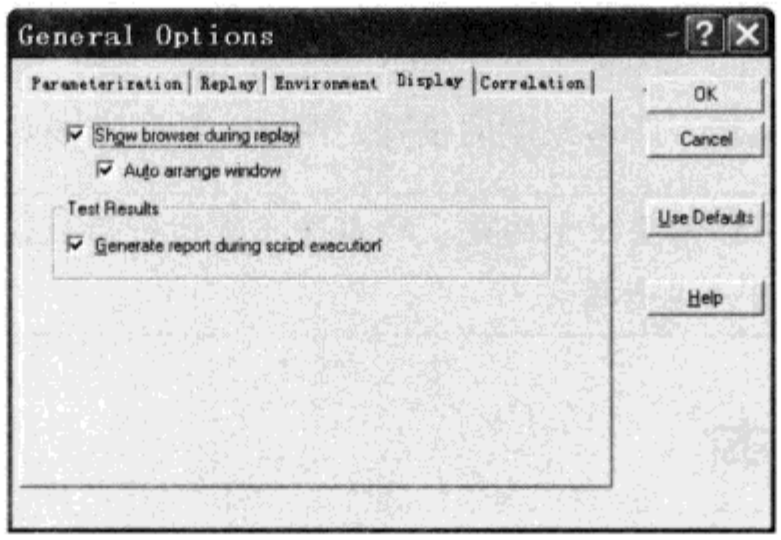


图 5-8 VU 开启运行时查看器



图 5-9 VU 运行时查看器

5.4 关联

当刚刚录制好的脚本回放不能成功时，你首先想到第一个可能出现的问题就是关联。

关联是 LoadRunner 中一个重要的应用，也是初学者经常犯错误的地方。在前面我们已经大概介绍了关联的原理，在本节则学习 LoadRunnerr 怎么做关联。

如果脚本需要关联（Correlation），在还没做之前是不会执行通过的，也就是说，会有错误信息发生。不过，很遗憾，并没有任何特定的错误信息是和关联（Correlation）有关系的。会出现什么错误信息，与系统实际的错误处理机制有关。错误信息有可能会提醒您要重新登录，但是也有可能直接就显示 HTTP 404 的错误信息。

这种问题在任何系统中都是非常常见的问题。在前面的章节我们已经了解了关联产生的背景和原理，其通用的解决模式是：“服务器返回给客户端一些动态变化的值，客户端使用这些值去访问服务器的时候，不能把这些值写死（hard-code）在脚本里面，而应该存放在一个变量里面。”这就是关联的概念。

关联的工作往往占据开发脚本的大部分时间，因为我们必须针对每一个具体的系统进行细致的分析，确定其需要关联的动态信息。而幸运的是，VU 为我们提供了三种关联机制。关联的方法有三种：

5.4.1　录制前 Correlation（关联）

我们在前面章节的有关关联原理中谈到，服务器就像一个饭店，而客户端就像一个拿着小票领饭的食客。如果我们在进饭店之前，已经预先知道小票的样子，那么这时我们就可以启用录制前关联了。录制前关联的原理是，我们在录制前建立关联的规则（提前告诉 VU 小票是什么样子），录制时 VU 按照这些规则寻找并建立关联（获得小票）。因此在 VU 中，录制前关联又叫做规则关联（Rule Correlation）。可见，使用录制前关联的必要条件是我们必须在录制脚本之前就知道哪些变量是需要关联的。

Vugen 内建自动关联引擎（auto-correlation engine），可以自动找出需要关联的值，并且自动使用关联函数建立关联。

在录制过程中 Vugen 会根据制定的规则，实时自动找出要关联的值。

1．规则的建立

关联的规则主要是指定两个边界，被关联变量的左边界和右边界，当 VU 在

Server 的 Respsonse 中找到符合条件的字符串时，它就意识到这是一张“小票”，赶紧把它保存下来，作为参数。

规则来源有两种：

（1）内建关联规则（Built-in Correlation）

Vugen 已经针对常用的一些应用系统，如 AribaBuyer、BlueMartini、BroadVision、InterStage、mySAP、NetDynamics、Oracle、PeopleSoft、Siebel、SilverJRunner 等内建关联规则，这些应用系统可能会有一种以上的关联规则。您可以在“Recording Options”>“HTTP Properties”>“Correlation”中启用关联规则（如图 5-10 所示），则当录制这些应用系统的脚本时，Vugen 会在脚本中自动建立关联。

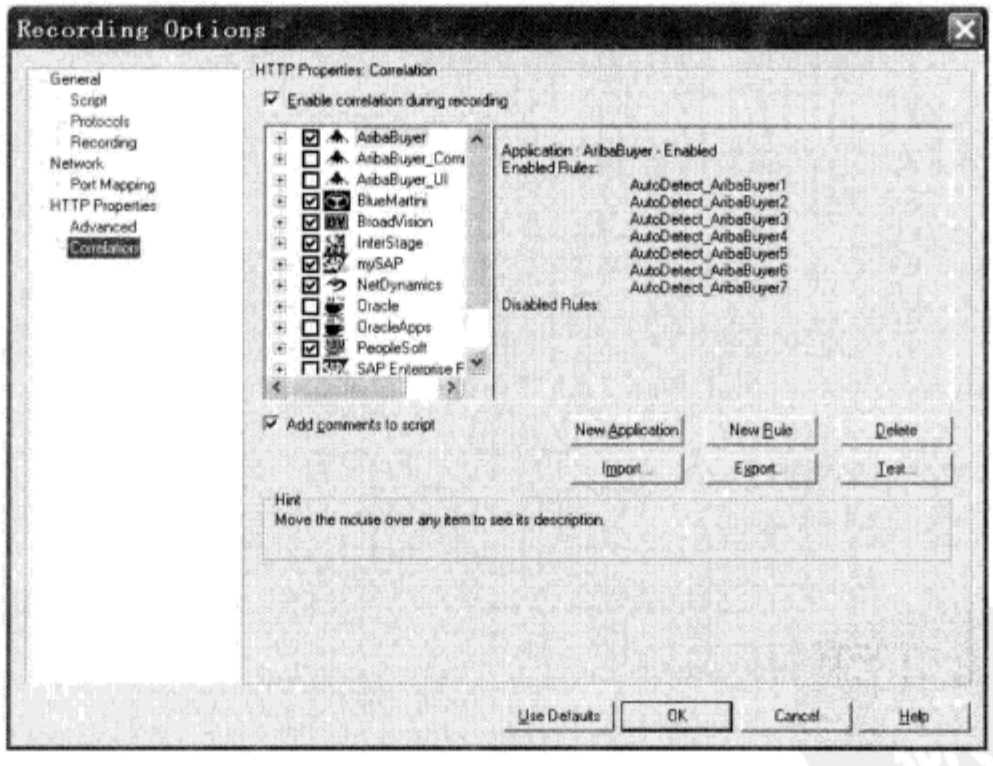


图 5-10 VU 内建关联规则

（2）使用者自订关联规则（User-defined Rules Correlation）

除了内建的关联规则之外，使用者也可以自订关联规则。您可以在“Recording Options”>“HTTP Properties”>“Correlation”中建立新的关联规则（如图 5-11 所示）。

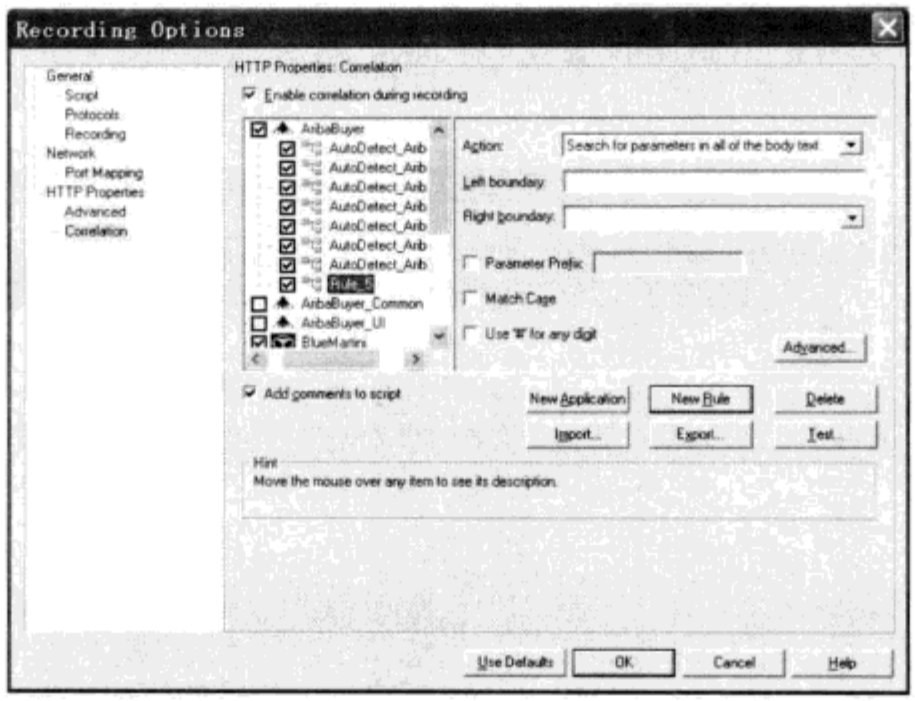


图 5-11　VU 创建一个新的关联规则

图 5-11 所示是创建新规则的页面，在创建新规则时，需要设置关联变量的左边界、右边界等信息。

2．规则的应用

依照以下步骤使用 Rule Correlation：

（1）启用自动关联（auto-correlation）

点选 Vugen 的“Tools”>“Recording Options”，打开“Recording Options”对话框，选取“HTTP Properties”>“Correlation”，勾选“Enable correlation during recording”，以启用自动关联。

假如录制的应用系统属于内建关联规则的系统，如 AribaBuyer、BlueMartini、BroadVision、InterStage、mySAP、NetDynamics、Oracle、PeopleSoft、Siebel、SilverJRunner 等，请勾选相对应的应用系统；或者也可以针对录制的应用系统加入新的关联规则，此即为使用者自订的关联规则。

（2）运行脚本验证关联机制

当关联规则都运行开始录制脚本时，在录制过程中，当 Vugen 侦测到符合关

联规则的数据时，会依照设定建立关联，您会在脚本中看到类似以下的脚本，如图 5-12 所示。



图 5-12　先建立关联规则，然后录制脚本

执行脚本验证关联是 OK 的。

5.4.2　录制后关联

当录制的应用系统不属于 Vugen 预设支持的应用系统时，也就是 VU 在录制之前并不知道服务器小票的样子，这时规则关联（Rule Correlation）就无能为力了，但不要紧，VU 还提供了一招，叫做录制后关联。

录制后关联：有别于内建关联，录制后关联则是在执行脚本后才会建立关联，也就是说，当录制完脚本后，脚本至少需被执行过一次，录制后关联才会作用。录制后关联会尝试找出录制时与执行时，服务器响应内容的差异部分，藉以找出需要关联的数据，并建立关联。

拿食客问题来解释录制后关联，就是食客第一次去某饭店吃饭时，他并不知道这个饭店的小票是什么版式，也不知道饭店给他一张写着“97”号的纸条是什么意思，不过不要紧，一回生，二回熟，第二次他去饭店，发现饭店又给他一张“108“号的纸条，他这时就能推断出，每次他去饭店这个号码都会变的，这应该就是饭店的小票。

因此使用录制后关联必须要让脚本运行第二次，步骤如下：

录制脚本并回放，回放完毕后，Vugen 会跳出下面的“Scan Action for Correlation”窗口，询问您是否要扫描脚本并建立关联，单击“Yes”按钮，如图 5-13 所示。

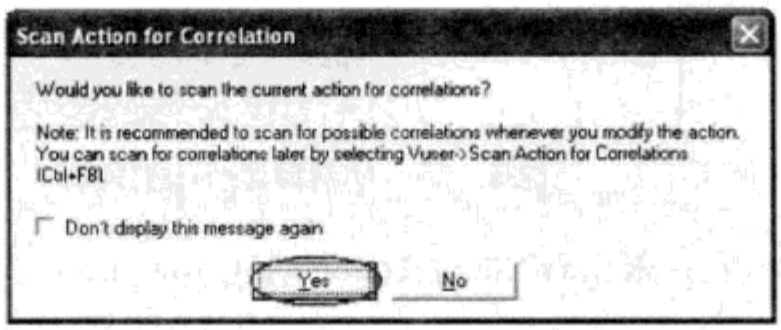


图 5-13　录制后关联提示页面

扫描完后，可以在脚本下方的“Correlation Results”中看到扫描的结果，如图 5-14 所示。

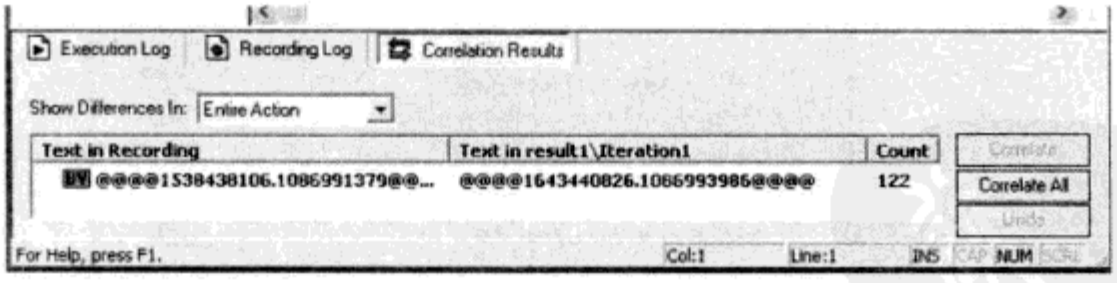


图 5-14　VU 的关联结果信息

检查一下扫描的结果，选择要做关联的数据，然后单击“Correlate”按钮，一笔一笔做，或者单击“Correlate All”按钮，让 Vugen 一次就对所有的数据建立关联。

注 意 由于录制后关联会找出所有有变动的数据，但是并不是所有的数据都需要做关联，所以不建议您直接用“Correlate All”。

要手动启动“Scan Action for Correlation”功能，请先执行脚本一次后，单击“Vuser”>“Scan Action for Correlations”，如图 5-15 所示。

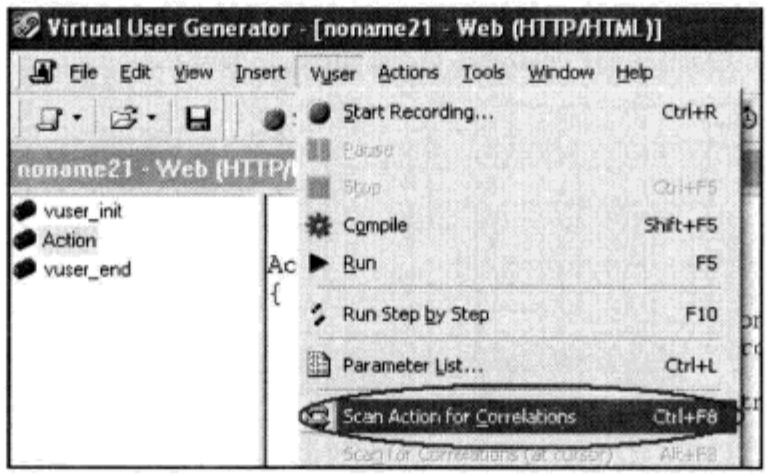


图 5-15 VU 选择运行扫描关联

执行完脚本后并未出现“Scan Action for Correlation”窗口，要启用“Scan Action for Correlation”功能，请单击“Tools”>“General Options”>“Correlation”选项卡，勾选“Show Scan for correlation popup after replay of Vuser”选项，如图 5-16 所示。

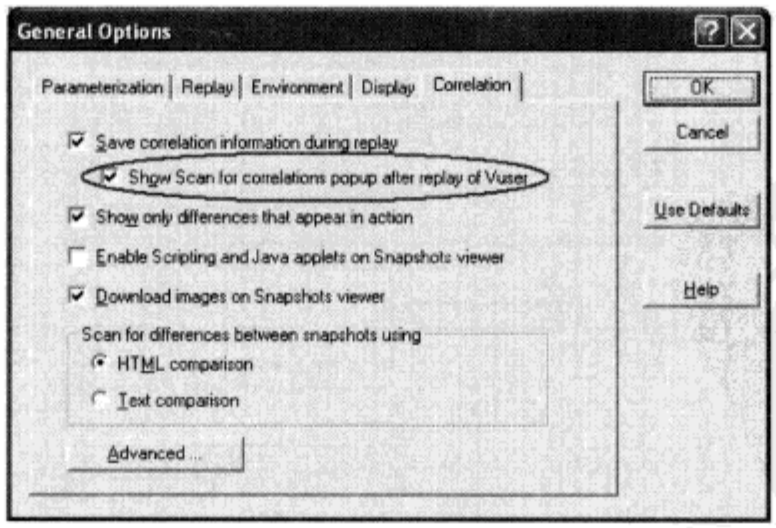


图 5-16 在 VU 选项中设定每次回放脚本弹出录制后关联提示框

一般来说，我们必须一直重复上面的步骤，直到所有需要做关联的数据都找

出来为止。因为有时前面的关联还没做好之前，将无法执行到后面需要做关联的部分。

有可能有些需要做关联的动态数据，连自动关联都无法侦测出来，这时您就需要自行做手动关联了。

5.4.3　手动关联

录制前关联和录制后关联应该都属于自动关联范畴。一般地，靠自动关联能够解决脚本中大多数的关联问题，但是万一它们都不能起到作用的话，那我们只能采用手动关联方法了。手动关联的原理和自动关联一样，就是要找出哪些变量是需要关联的，然后用 LoadRunner 提供的 web_reg_save_param 来实现关联。

手动关联的执行过程大致如下：

（1）使用相同的业务流程与数据，录制两份脚本。

> **注意**　一定要保证两次录制的步骤和输入都完全一样，因为理论上，如果两次客户端的请求完全一样，服务器的返回结果就应该是一样的。但是由于小票情形的存在，我们就可以大致推断两次服务器返回结果的不一致处极有可能就是需要关联的变量。最常见的就是 Java Web 应用的 jsessionid 等数据。

（2）使用 WinDiff 工具协助找出需要关联的数据。

WinDiff 是内嵌在 VU 中的一个文件比较工具，用于比较两个文件的内容。我们在这里用它来比较两个脚本的不同之处，从而找到需要被关联的数据。对了，刚才说到被关联的数据本来是服务器返回的，我们为什么要比较脚本呢？因为服务器的小票最终是要被客户端再次使用的，只要客户端一使用小票，那么就会被 WinDiff 发现。

我们可以在“Tools”菜单下选择“Compare with Vuser”选项，如图 5-17 所示。

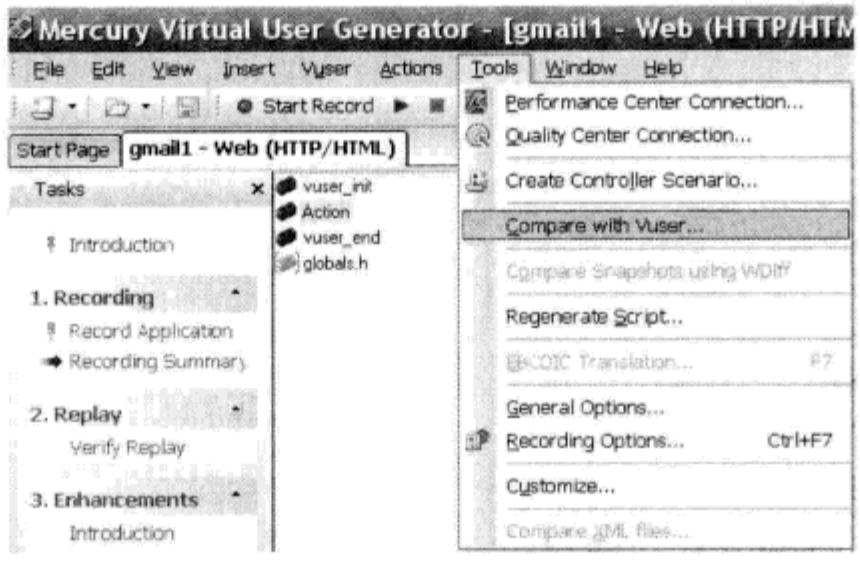


图 5-17 在 VU 的 Tools 菜单中选择“Compare with Vuser”选项

弹出如图 5-18 所示的对话框。

WinDiff 要比较的是两个脚本，那么当前脚本就是第一个脚本，而在上面的浏览对话框里，我们选择的是第二个脚本路径。

（3）确定被关联数据的左、右边界。

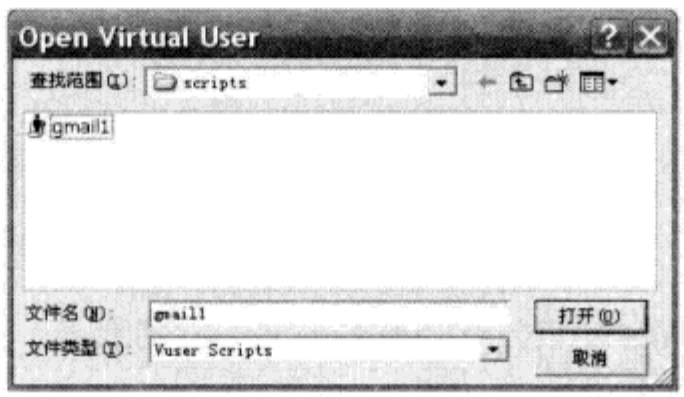


图 5-18 选择要比较的另外一个脚本

注 意 要确定关联数据的左、右边界，必须打开扩展日志。

在如图 5-19 所示的对话框中，在“Log options”中选中“Extended log”，然后至少要勾选“Data returned by server”一项。

这时再回放脚本，回放日志里记录的就是一个完整的客户端到服务器的交互内容，在回放日志里查找此关联数据，注意 Response body 字样的数据段，这标明它后面的内容是从 Server 返回的。

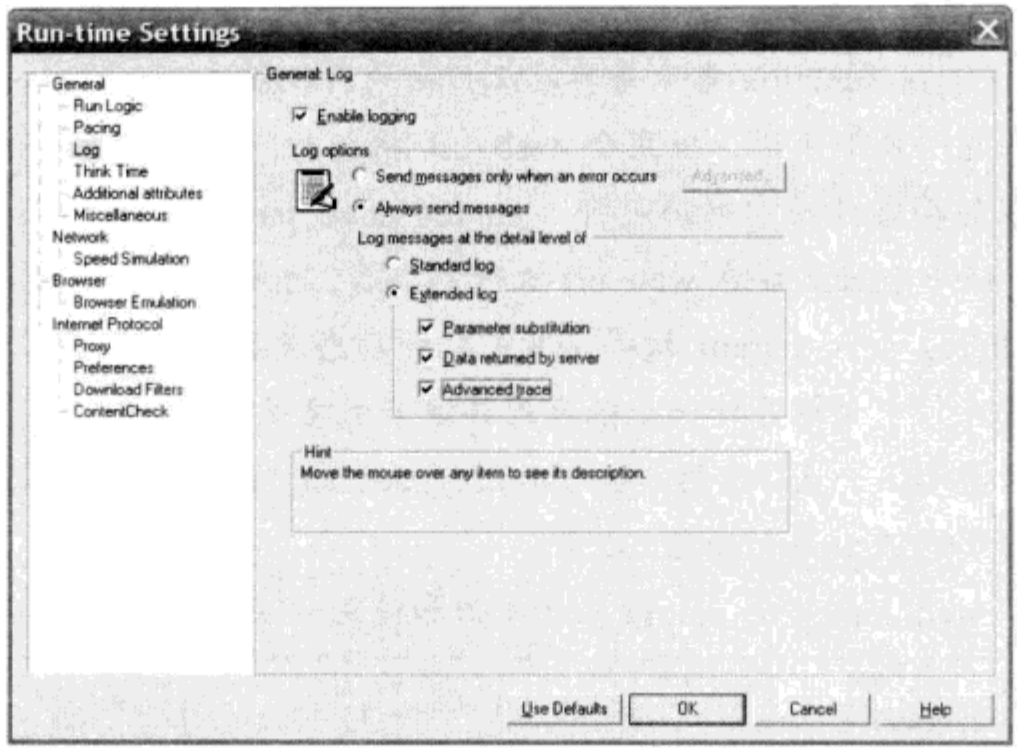


图 5-19　在 Run-time settings 中设置开启扩展日志

（4）使用 web_reg_save_param 函数手动建立关联。

（5）将脚本中有用到关联的数据，以参数取代。

关于整个手动关联的详细过程，有兴趣的朋友可以在互联网上 google 之，在此不再占用篇幅赘述。



关于 web_reg_save_param 函数

对于关联（Correlation）来说，web_reg_save_param 是最重要的一个函数，其功能是在下载的网页内容中，通过设定的边界字符串，找出特定的数据并将其储存在一个参数中，以供后续脚本使用。

接下来将针对 web_reg_save_param 做比较详细的说明。

web_reg_save_param 是一个 Service function。Service function 主要用来完成一些特殊的工作，如关联、设定 Proxy、提供认证信息等，当其作用时，不会对网页的内容做任何的修改。

web_reg_save_param 同时也是一个 registration type function（注册型函数）

（只要函数名称中包含“_reg_”字眼，就表示其为 registration type function）。registration type function 意味着其真正作用的时机是在下一个 Action function 完成时执行的。举例来说，当某个 web_url 执行时所接收到的网页内容中包含了要做关联的动态数据，则必须将 web_reg_save_param 放在此 web_url 之前，web_reg_save_param 会在 web_url 执行完毕后，也就是网页内容都下载完后，再执行 web_reg_save_param 找寻要做关联的动态数据并建立参数。我们会在 5.8 节“检查点”中介绍另外一个很有用的注册类型函数：web_reg_find。

所以要记住一点，使用 registration type function 时，要注意其放置的位置必须在要作用的 Action function 之前。

关于 web_reg_save_param 函数手册和使用方法可参看 LoadRunner 手册。



Loadrunner 8.0 中的快照（snapshot）功能得到进一步的增强，通过 snapshot 可以更加直观地查看客户端的请求，服务器的回应数据，也更加方便了关联功能。

综上所述，关联的用途和意义如下：

（1）简化和优化脚本代码

比如，需要执行一系列有关联的查询，以一个查询结果作为另外一个查询的条件，这样可能使脚本代码很长；如果通过嵌套循环的方法来实现，又会降低代码的可读性，难以理解，而关联就可解决这个问题。

（2）动态产生数据

这常见于许多网站应用。很多网站用当前系统时间戳来标志 Session。显然，先录制，后回放，原先生成的时间戳下次就不能用了，因此关联能够每次更新时间戳。

（3）支持唯一数据

有些系统要求使用唯一的数据。录制时生成的数据，在回放时就重复了，无

法保持唯一，也将不可用。例如，假设你录制创建一个账户的过程，每个账户的 ID 都是唯一的，对于用户是不可见的。这些账户 ID 被用作 primary key 被插入数据库表中，这样的话，回放时就会失败。使用关联就可以解决这个问题。

手动关联实现的原理是：

（1）捕获第一个操作的输出

这里首先知道哪个值是需要关联的。可以用 wdiff（LoadRunner 自带）去对比两个录制同样操作的脚本，其中不同的地方可能就是需要关联的。

（2）保存为参数

在上面的例子中，把查询结果保存成 LoadRunner 的一个变量。不同的 Vuser 有不同的实现函数。通常函数名中包含 save_param，比如 Web Vuser 中是 web_reg_save_param、lrs_save_param 等。

（3）将参数传递到另外一个操作的输入

即把另外一个操作中的常数，替换成参数变量。

5.5　脚本视图和树视图

VU 提供两种视图来查看脚本的内容，一个是脚本视图，另一个是基于图标的树视图（内有快照）。

所有类型的 Vuser 都有文本脚本视图，但是只有特定的 Vuser 才会有树视图。

5.5.1　树视图（Tree View）

Tree View 也叫做基于 icon 的 View，也就是说，脚本的每个函数在 Tree View 中都以一个带有 icon 的节点来代替。可以点击工具栏中的“Tree”按钮或者在“View”菜单下选择“Tree View”，显示 VU 树视图，如图 5-20 所示。

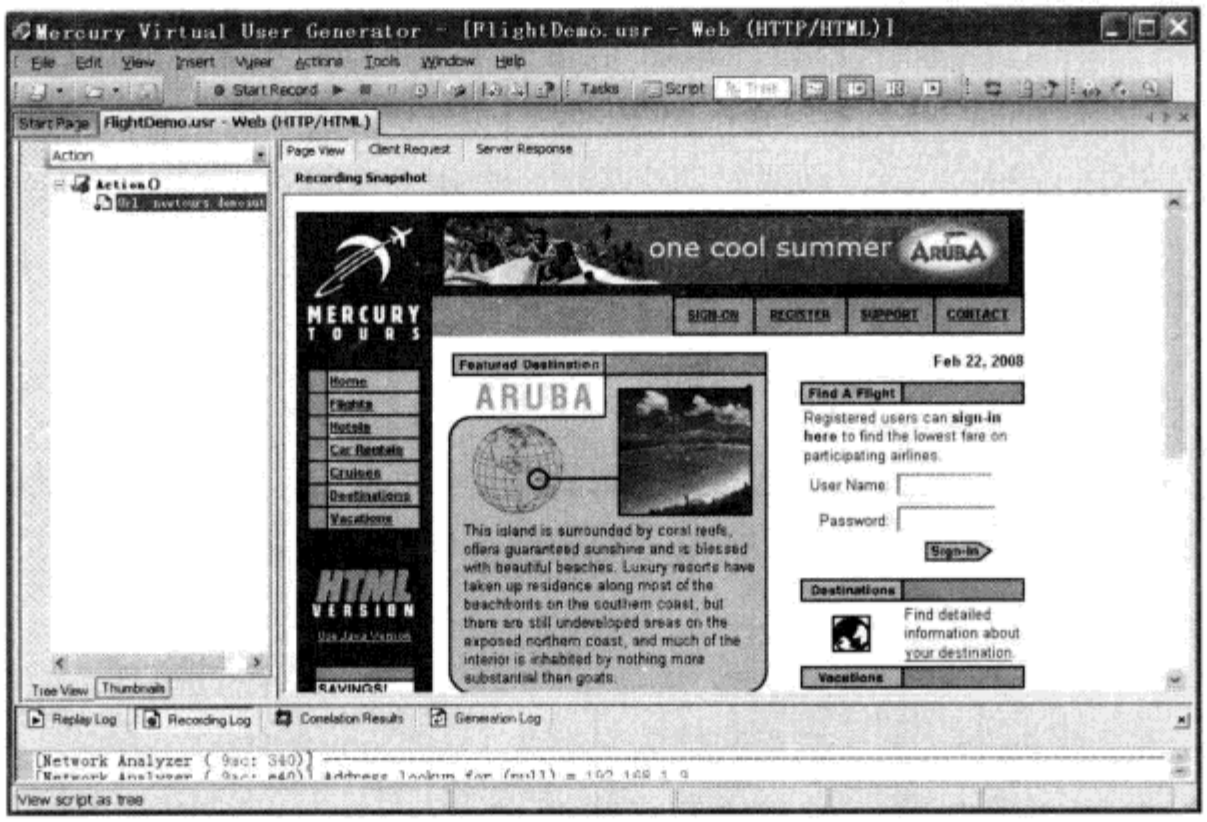


图 5-20 VU 树视图

Tree View 的好处是使用户更方便地修改脚本，Tree View 支持拖拽，用户可以把任意一个节点拖拽到他想要的地方，从而达到修改脚本的目的。用户可以右键单击节点，进行修改/删除当前函数参数属性，增加函数等操作，通过 Tree View 能够增加 LoadRunner 提供的部分常用通用函数和协议相关函数。比如 Web Service Vuser 就不能通过 Tree View 参数化一些复杂的数据类型，在这种情况下，就需要 Script View 了。

5.5.2 脚本视图（Script View）

在 Script View 中能够看到一行行的 API 函数，Script View 适合一些高级用户，通过 Script View 向脚本中增加一些其他 API 函数。可以单击工具栏上的“Script”按钮或者在“View”菜单下选择“Script View”，显示 VU 脚本视图，如图 5-21 所示。

注 意 当用户在 Script View 中对脚本做了修改之后，Tree View 也会做相应的变化。如果脚本有语法错误，Script View 将不能转化为 Tree View 或缩略图。



图 5-21　VU 脚本视图

5.5.3　理解 Snapshot

Snapshot，顾名思义，就是快照，代表当前的 step，Snapshot 显示了客户端在执行完当前 step 后的样子。在 Tree Ciew 右侧的 frame 中可以查看 Snapshot，在 LoadRunner 8.0 中，Snapshot 包含 Page View、Client Request 和 Server Response。Snapshot 有两种生成方式，一种是在 record 的时候生成，另一种是在 replay 的时候生成。你可以对比两种方式生成的 Snapshot，以发现哪些是动态值，需要参数化。

5.6　事务、同步点和思考时间

5.6.1　Transaction（事务）

事务是计算机程序设计中一个很重要的概念。一个事务应该具有原子性、一致性、隔离性和持久性。这 4 个属性的详细解释在网上都可以找到。在 LoadRunner 里，我们定义事务主要是为了度量服务器的性能。每个事务度量服务器响应指定

的 Vuser 请求所用的时间，这些请求可以是简单任务（例如等待对单个查询的响应），也可以是复杂任务（例如提交多个查询和生成报告）。

要度量事务，需要插入 Vuser 函数以标记任务的开始和结束。在脚本内，可以标记的事务不受数量限制，每个事务的名称都不同。

在场景执行期间，Controller 将度量执行每个事务所用的时间。场景运行后，可使用 LoadRunner 的图和报告来分析各个事务的服务器性能。

设置 Transaction 的方法如下：

- 选择新 Transaction 开始点，在被度量脚本段之前插入 lr_start_transaction。
- 选择新 Transaction 结束点，在被度量脚本段之后插入 lr_end_transaction。

下面的脚本例子中将登录操作设为一个名为“login”的事务：

```
Lr_start_transaction("login");
web_submit_form("auth",
        "Snapshot=t2.inf",
        ITEMDATA,
        "Name=ssousername",
        "Value=robin", ENDITEM,
        "Name=password", "Value=123456", ENDITEM,
        "Name=remember", "Value=<OFF>", ENDITEM,
        LAST);
Lr_end_transaction("login");
```

如果上面手工插入Transaction函数看作是“显式事务”的话，那么LoadRunner还提供了一种“隐式事务”的机制，在 VU 的 Run-time Settings 中又称为“自动事务”。

在 Run-time Settings 中，在 Miscellaneous 选项卡的 Automatic Transactions 中定义自动事务。

可以设置 LoadRunner 直接按事务处理 Vuser 中的每个 Action 或 step。这里，Action 指的是 vuser_init、Action 和 vuser_end 三大函数，而 step 指的是 LoadRunner

执行的每个函数。LoadRunner 将 Action 名或 step 名指定为事务名。在默认情况下，将启用按 Action 使用自动事务的功能，如图 5-22 所示。

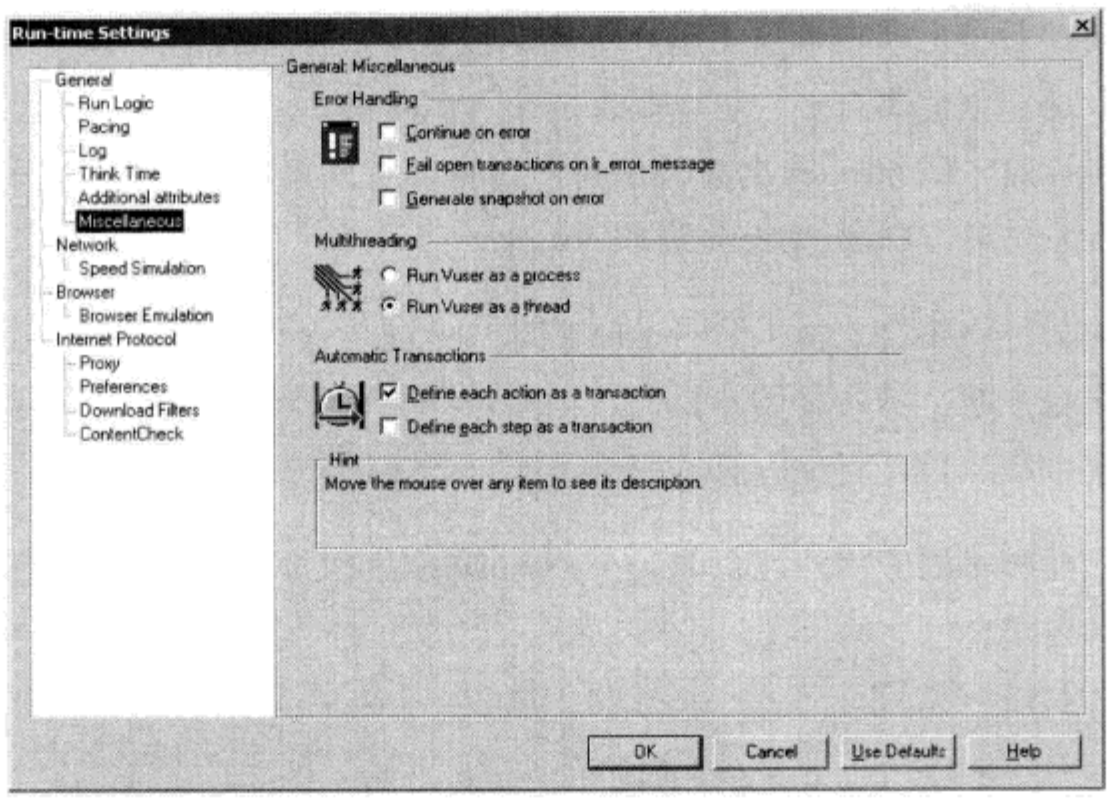


图 5-22　VU 定义自动 Transaction

- 要禁用按操作使用自动事务的功能，清除“Define each action as a transaction”复选框（默认情况下启用）。
- 要启用按步骤使用自动事务的功能，选中“Define each step as a transaction”复选框（默认情况下禁用）。



Transaction 的开始点和结束点必须在一个 Action 中，跨越多个 Action 是不允许的。

Transaction 的名字在脚本中必须是唯一的，当然也包括在多 Action 的脚本中。

你也可以在一个 Transaction 中创建另外一个 Transaction，叫做 Nested Transaction。详细使用方法可参看 LoadRunner 函数手册。

5.6.2 Rendezvous Point（同步点）

要在系统上模拟较重的用户负载，需要同步各个 Vuser 以便在同一时刻执行任务。通过创建集合点，可以确保多个 Vuser 同时执行操作。当某个 Vuser 到达该集合点时，Controller 会将其保留，直到参与该集合的全部 Vuser 都到达。当满足集合条件时，Controller 将释放 Vuser。

可通过将集合点插入到 Vuser 脚本中来指定会合位置。在 Vuser 执行脚本并遇到集合点时，脚本将暂停执行，Vuser 将等待 Controller 允许继续执行。Vuser 被从集合释放后，将执行脚本中的下一个任务。

下面脚本以多用户运行时（在 Controller 中执行场景），所有的用户会一同发起登录请求。

```
Lr_rendezvous("the begin port");
web_submit_form("auth",
        "Snapshot=t2.inf",
        ITEMDATA,
      "Nhttp://bbs.51testing.com/thread-106686-1-1.html
"Value=<OFF>", ENDITEM,
        LAST);
```



只能在 Action 中添加集合点（不能在 vuser_init/vuser_end 中添加）。



因为同步点是协调多个虚拟用户的并发操作，显而易见，在 VU 运行脚本时，同步点是没有任何意义的，只有 Controller 多用户并发场景时，同步点的意义才表现出来（我们在 Controller 一章中会有详述）。

? 思考

在 Transaction 的开始和结束中间，插入同步点会对 Transaction 有什么影响？

5.6.3　注释

写脚本和写程序一样，应该养成经常写注释的习惯。

在 LoadRunner C 脚本中，LoadRunner 支持 C 的注释方法。

以下是文件头注释，如图 5-23 所示。

```
/*
Name: Login fligt demo
Description:Access the indexpage, then login with testing/123456
Author: Shining
Created: 2008-01-01
History:
    Modified on 2008-01-02 For parameterize
*/
Action()
{
    return 0;
}
```

图 5-23　VU 脚本信息注释

也可插入注释以描述脚本函数功能的信息，如图 5-24 所示。

```
Action()
{

    //Access the index page http://newtours.demoaut.com
    web_url("newtours.demoaut.com",
        "URL=http://newtours.demoaut.com/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t24.inf",
        "Mode=HTML",
        LAST);
    //Login with testing/123456
    web_submit_form("login.php",
        "Snapshot=t25.inf",
        ITEMDATA,
        "Name=userName", "Value=testing", ENDITEM,
        "Name=password", "Value=123456", ENDITEM,
        "Name=login.x", "Value=0", ENDITEM,
        "Name=login.y", "Value=0", ENDITEM,
        LAST);

    return 0;
}
```

图 5-24　VU 脚本函数注释

5.6.4　Think Time（思考时间）

用户在执行两个连续操作期间等待的时间称为“思考时间”。Vuser 使用 lr_think_time 函数模拟用户思考时间。录制 Vuser 脚本时，Vugen 将录制实际

的思考时间并将相应的 lr_think_time 语句插入到 Vuser 脚本。可以编辑已录制的 lr_think_time 语句，而且可以向 Vuser 脚本中手动添加更多的 lr_think_time 语句。



提示 添加以上语句的途径有多种，我们可以通过 Tree View、Script View 以及快捷图标来添加事务、同步点、思考时间和注释等函数。



提示 lr_think_time 的参数单位是秒，比如 lr_think_tim(5)意味着 LoadRunner 执行到此条语句时，停留 5 秒，然后再继续执行后面的语句。



提示 如果不想在脚本中执行 Think Time 语句，需要逐条语句删除，那是件很麻烦的事情，所以 LoadRunner 提供了在 Run-time Settings 中可以设置直接忽略 Think Time，而不用修改脚本，如图 5-25 所示。

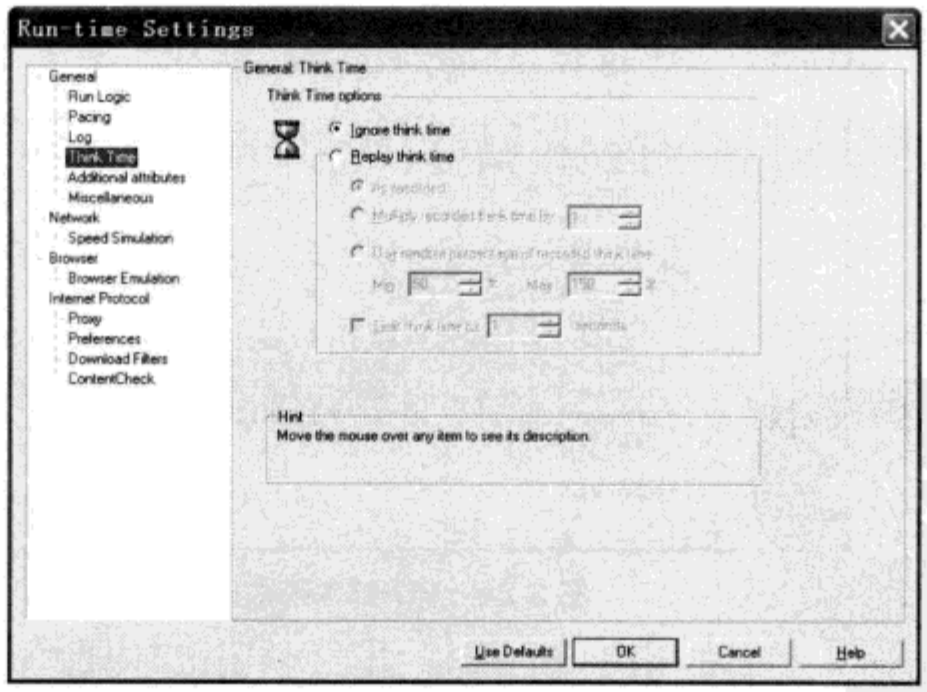


图 5-25 在 VU 运行时设置中忽略 Think Time

上述设置生效后，脚本运行时，脚本中所有的 lr_think_time 语句都不会被执行。

思考

在一个 Transaction 里，插入 lr_think_time 语句，对 Transaction 会有什么影响？

思考

有 Think Time 的脚本和没有 Think Time 的脚本相比，哪个给服务器造成的压力大？哪个更符合用户实际的工作场景？

5.7　数据驱动——参数化（Parameters）

数据驱动就是把测试脚本和测试数据分离开来的一种思想，脚本体现测试流程，数据体现测试案例。数据不是 hard-code 在脚本里面，这样大大提高了脚本的可复用性。而 LoadRunner 的参数化功能是数据驱动测试思想的一个重要实现。

在本节中，我们要学习的是：理解参数的局限性，建立参数，定义参数的属性，理解参数的类型，为局部数据类型设置参数的属性，为数据文件设置参数的属性，从已经存在的数据库中引入数据。

5.7.1　为什么需要参数化

在录制程序运行的过程中，Vugen（脚本生成器）自动生成了脚本以及录制过程中实际用到的数据。在这个时候，脚本和数据是混在一起的。

比如，你用 VU 的 Web Vuser 录制一个用户登录 Web 系统的过程，对于登录的操作，会生成以下脚本：

```
web_submit_form("auth",
        "Snapshot=t2.inf",
        ITEMDATA,
        "Name=ssousername",
        "Value=robin", ENDITEM,
        "Name=password", "Value=123456", ENDITEM,
        "Name=remember", "Value=<OFF>", ENDITEM,
        LAST);
```

web_submit_form 是登录触发的动作，而“robin”和“123456”是填入的数据。如果 Controller 里以多用户方式运行这个脚本的时候，每个虚拟用户都会以同样的用户名“robin”、密码“123456”去登录 Web 系统。这样做性能测试，我们的客户可能不会答应，因为这显然不是一个真实的业务场景。尤其现在服务器大多会采用 Cache 功能提高系统性能，用同样的用户名/密码登录系统的 Cache 命中率会很高，也要快得多。

因此，我们的客户希望当用 LoadRunner 多用户多循环运行时，不会只是重复一个用户的登录，也就是说，此函数中的数据要能变化，这样的话，就把这些数据用一个参数来代替，其实就是把常量变成变量。参数化后的脚本如下：

```
web_submit_form("auth",
        "Snapshot=t2.inf",
        ITEMDATA,
        "Name=ssousername",
        "Value={username}", ENDITEM,
        "Name=password",
        "Value={passwd}", ENDITEM,
        "Name=remember", "Value=<OFF>", ENDITEM,
        LAST);
```

参数化后，用户名“robin”被一个参数{username}替换，密码“123456”被另外一个参数{passwd}代替。{username}和{passwd}分别和参数文件关联，在脚本运行时，用户名和密码的值从参数{username}和{passwd}中获得。而我们会在后面介绍 LoadRunner 有一套机制来保证参数的使用和变化，这样就实现了脚本与数据的分离。

参数化是我们学习 LoadRunner 中经常用到的功能。除了实现数据驱动之外，参数化脚本还有以下两个优点：

（1）可以使脚本的长度变短。

（2）可以增强脚本的可读性和可维护性。

实际上，参数化的过程如下：

（1）在脚本中用参数取代常量值。

（2）设置参数的属性以及数据源。

这些我们会在下面内容中详细介绍。

5.7.2　参数的创建

LoadRunner 对脚本中参数个数没有限制，我们可以在一个脚本中创建任意多个参数。

下面以 Web Vuser 为例，看看 LoadRunner 是如何创建参数的。我们已经知道，VU 可以通过 Tree View 和 Script View 两种途径来改变脚本，包括参数化功能。我们这里分别介绍。

还是上面那个登录脚本，我们可以在基于文本的脚本视图中参数化。

1．脚本视图参数化

① 将光标定位在要参数化的字符上，单击右键，弹出快捷菜单，如图 5-26 所示。

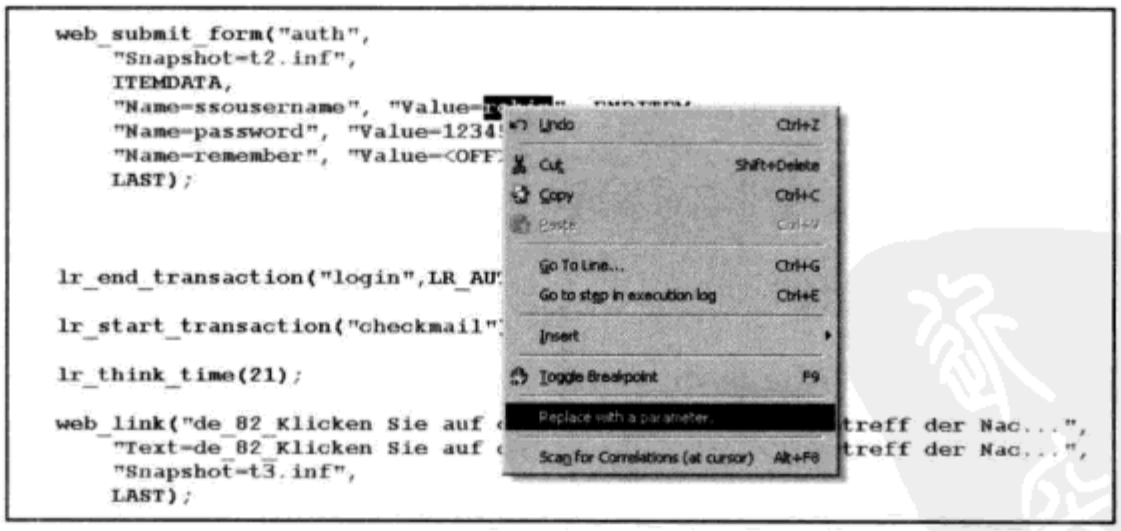


图 5-26　脚本参数化之右键选择替代参数

② 在弹出菜单中，选择“Replace with a Parameter”，打开选择或者创建参数对话框，如图 5-27 所示。

③ 在“Parameter name”中输入参数的名称，或者选择一个在参数列表中已经存在的参数。

④ 在“Parameter type”下拉列表中选择参数类型，如图 5-28 所示。

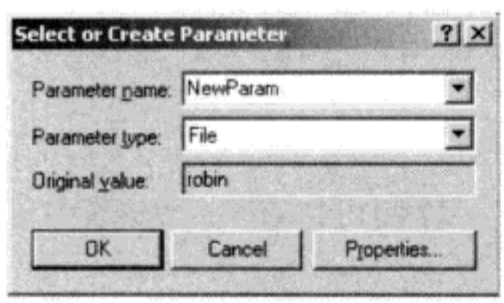


图 5-27 脚本参数化之设定参数名字

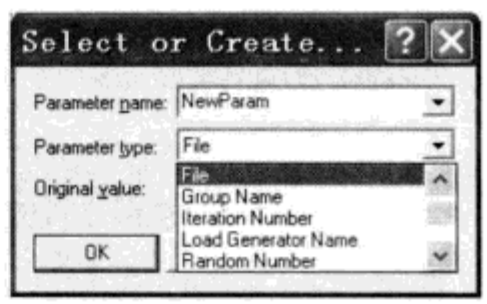


图 5-28 脚本参数化之设定参数类型

下面重点介绍一下参数的类型。

在定义参数属性的时候，要指定参数值的数据源。你可以指定下列数据源类型中的任何一种：

（1）Data Files

这是我们最常使用的一种参数类型，它的数据存在于文件中。该文件的内容可以手工添加，也可以利用 LoadRunner 的 Data Wizard 从数据库中导出。我们将在后面详细地介绍。

（2）User-Defined Functions

调用外部 DLL 函数生成的数据。

（3）Internal Data

虚拟用户内部产生的数据。

Internal Data 包括以下几种类型：

- **Date/Time**

Date/Time 用当前的日期/时间替换参数。要指定一个 Date/Time 格式，你可以从菜单列表中选择格式，或者指定自己的格式。这个格式应该和脚本中录制的

Date/Time 格式保持一致。

- **Group Name**

Group Name 用虚拟用户组名称替换参数。在创建 scenario 的时候，你可以指定虚拟用户组的名称。注意：当从 VU 运行脚本的时候，虚拟用户组名称总是 None。

- **Load Generator Name**

Load Generator Name 用脚本负载生成器的名称替换参数。负载生成器是虚拟用户在运行的计算机。

- **Iteration Number**

Iteration Number 用当前的迭代数目替换参数。

- **Random Number**

Random Number 用一个随机数替换参数。通过指定最大值和最小值来设置随机数的范围。

- **Unique Number**

Unique Number 用一个唯一的数字来替换参数。你可以指定一个起始数字和一个块的大小。



注 意　使用该参数类型必须注意可以接受的最大数。例如：某个文本框能接受的最大数为 99。当使用该参数类型时，设置第一个数为 1，递增的数为 1，但 100 个虚拟用户同时运行时，第 100 个虚拟用户输入的将是 100，这样脚本运行将会出错。

- **Vuser ID**

Vuser ID 用分配给虚拟用户的 ID 替换参数，ID 是由 LoadRunner 的控制器在

scenario 运行时生成的。如果从脚本生成器运行脚本的话，虚拟用户的 ID 总是-1。

⑤ 我们给此参数命名为“username”，然后单击“OK”按钮，关闭该对话框。脚本生成器便会用参数中的值来取代脚本中被参数化的字符，参数用一对“{}”括住，如图 5-29 所示。

```
web_submit_form("auth",
    "Snapshot=t2.inf",
    ITEMDATA,
    "Name=ssousername", "Value={username}", ENDITEM,
    "Name=password", "Value=123456", ENDITEM,
    "Name=remember", "Value=<OFF>", ENDITEM,
    LAST);
```

图 5-29 脚本参数化后的变化



提示 在参数化 CORBA 或者 General-Java 用户脚本的时候，必须参数化整个字符串，而不是其中的部分。另外注意：除了 Web 或者 WAP，缺省的参数括号对于任何脚本都是“{}”。你可以在“General Options”对话框的“Parameterization”选项卡（“Tools”>“General Options”）中定义参数括号种类。

⑥ 用同样的参数替换字符的其余情况。选中参数，单击右键，在弹出的菜单中选择“Replace more occurrences”，如图 5-30 所示，打开搜索和替换对话框。“Find What”中显示了你企图替换的值，“Replace With”中显示了括号中参数的名称。选择适当的检验框来匹配整个字符或者大小写。如果要搜索规则的表达式（.、！、？等），则选中“Regular Expression”检验框，然后单击“Replace”或者“Replace All”按钮。



提示 小心使用“Replace All”，尤其替换数字字符串的时候。脚本生成器将会替换字符出现的所有情况。

⑦ 如果想用以前定义过的参数来替换常量字符串的话，选中该字符串，单击右键，然后选择“Use existing parameters”，从弹出的子菜单中选择参数，或者用“Select from Parameter List”来打开参数列表对话框，如图 5-31 所示。

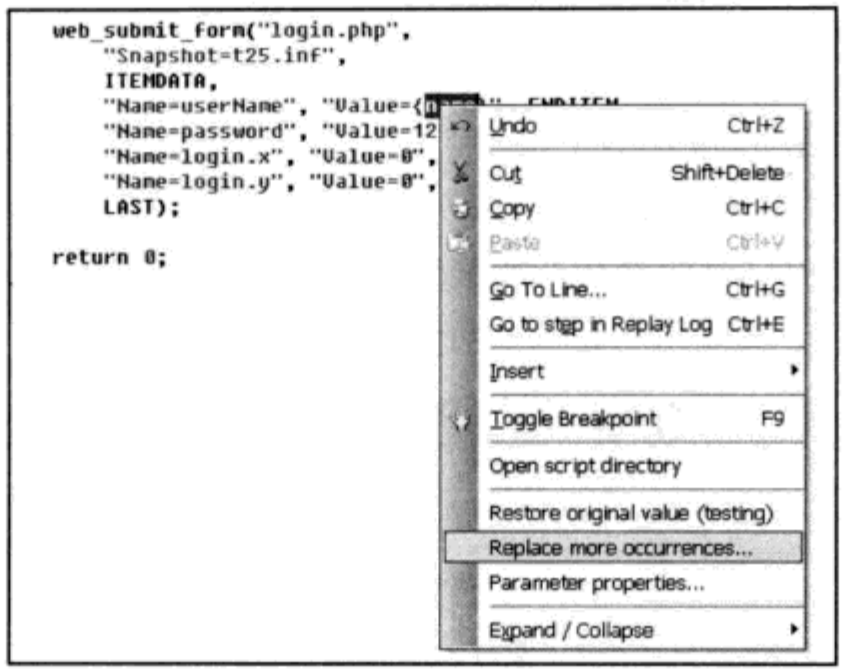


图 5-30　在右键菜单中选择替换其他变量

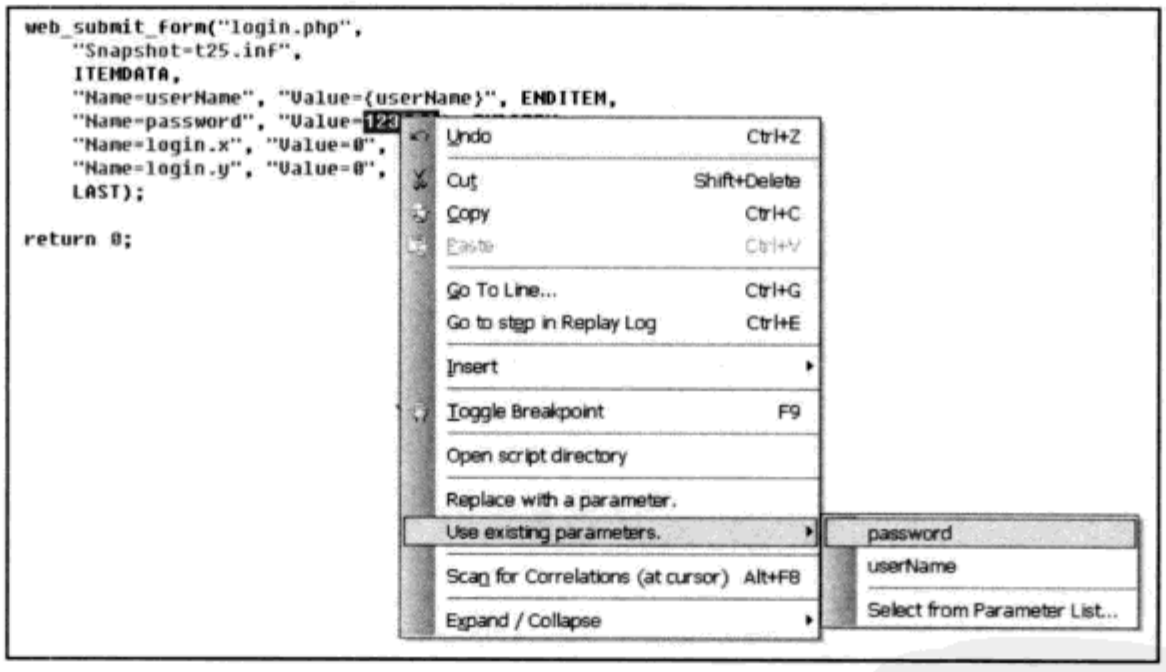


图 5-31　在右键菜单中选择使用已有参数



提示　如果用以前定义过的参数来替换常量字符串的话，那么，使用“Parameter List”非常方便。同时，还可以查看和修改该参数的属性。

⑧ 对于已经用参数替换过的地方，如果想取回原来的值，那么就在参数上单击右键，然后选择“Restore Original value”。

不是所有的数据都可以参数化。

上面 LoadRunner 给我们提供了一种很方便的机制去参数化。但这种机制的应用范围是有限的，只有函数的参数才能参数化，不能参数化非函数参数的数据。另外，不是所有函数的参数都能参数化。可以通过查看 loadrunner 自带的函数手册来确认是否可参数化，如图 5-32 所示。

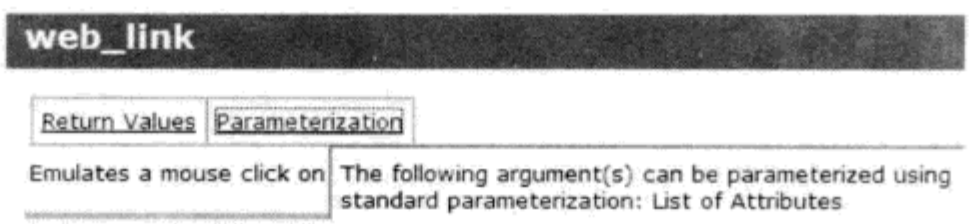


图 5-32　函数入口参数化

例如：

```
    lrd_stmt (LRD_CURSOR FAR *mptCursor, char FAR *mpcText, long
mliTextLen, LRDOS_INT4 mjOpt1, LRDOS_INT4 mjOpt2, int miDBError
Severity);
```

LoadRunner 只支持 mpctext 参数化：

```
    lrd_stmt(Csr4, "select name from sysobjects where name =\"<name>\"
", -1, 148, -99999, 0);
```

那么，对于不能使用上面机制参数化的数据，我们有没有其他办法将其参数化呢？

答案是 lr_eval_string，我们可以在 Vuser 脚本中的任何地方使用 lr_eval_string 来参数化数据。lr_eval_string 用来得到一个参数的值，而参数可以预先在 LoadRunner 的 Parameter List 里定义好，也可以是之前通过其他函数创建的。下面是一个小例子：

```
    //通过 lr_save_datetime 把七天后的时间保存在 date 参数中
    lr_save_datetime("%d %m %y", DATE_NOW + (ONE_DAY*7), "date");
    //通过 lr_eval_string 把 date 参数中的值取出来，lr_output_message 的输
出值为七天后的时间
    lr_output_message("Date is %s",lr_eval_string("{date}"));
```

其详细使用方法可参见 LoadRunner 函数手册。这里需要注意的是：lr_eval_string 函数的返回值是一个指向参数值的指针，这个指针指向的内存是 LoadRunner 内部分配的，每次 Iteration 后自动释放。如果在 Iteration 中还有多层循环进行参数化，那么最好不要使用 lr_eval_string 了，这会导致内存迟迟不能释放。在这种情况下，应该使用 lr_eval_string_ext，同时配对使用 lr_eval_string_ext_free 来及时释放内存。

2. 在 Web 用户脚本的树视图中创建参数

我们在前面说到 VU 脚本的树视图和脚本视图其实是同一个脚本的两个展现方式而已，那么在树视图中，我们怎样做参数化呢？

① 将光标定位在要参数化的地方，单击右键，从弹出的菜单中选择“Properties”，如图 5-33 所示，则打开相关的属性对话框。

② 单击要参数化的参量旁边的“ABC”形状的图标，打开“Select or Create Parameter”对话框，如图 5-34 所示。

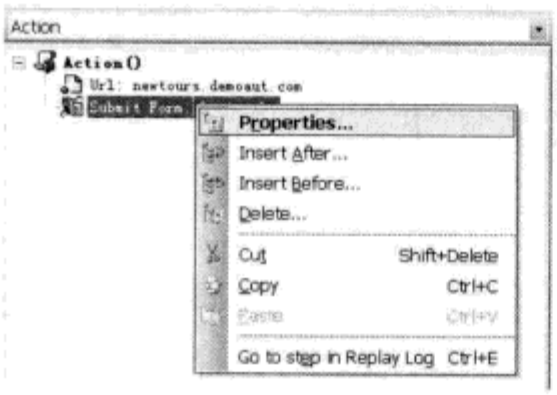


图 5-33　在树视图中选择右键菜单中的属性

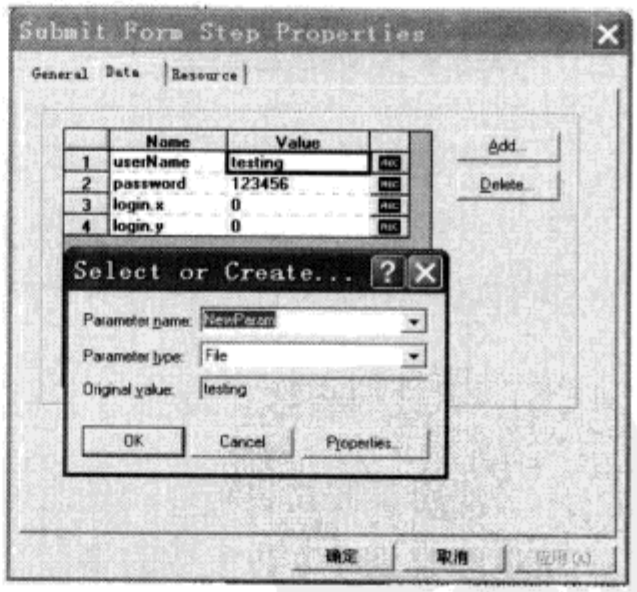


图 5-34　在树视图中创建新参数

③ 在“Parameter name”中输入参数的名称，或者从列表中选择一个已经存在的参数。

④ 在“Parameter type”中输入参数的类型。

⑤ 单击“OK”按钮关闭该对话框。用户脚本生成器会用参数来替换最初的字符串常量，并用一个表格形状的图标替换“ABC”形状的图标，如图 5-35 所示。

⑥ 要恢复参数化以前的值，单击图标，然后从弹出的菜单中选择“Undo Parameter”，如图 5-36 所示，则以前的值便会重现。

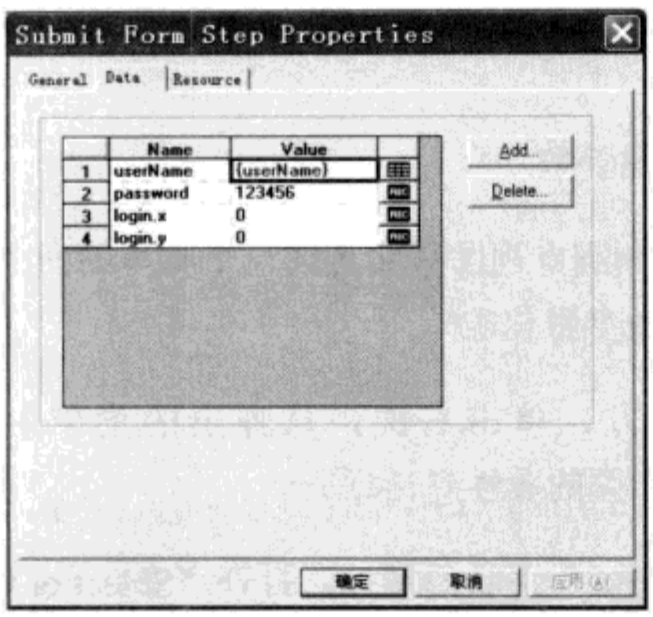


图 5-35 在树视图中成功创建参数

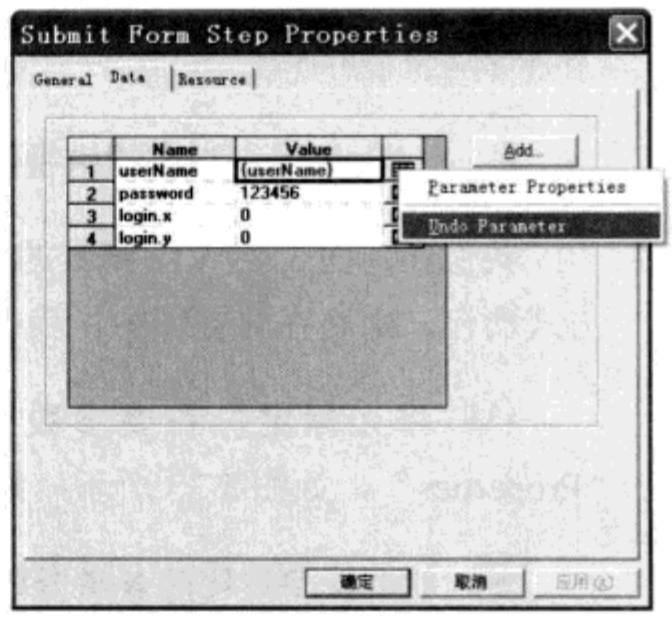


图 5-36 在树视图中恢复参数

5.7.3 定义参数的属性

创建参数完成后，就可以定义其属性了。参数的属性定义就是在脚本执行过程中，定义参数使用的数据源。在 Web 用户脚本中，既可以在基于文本的脚本视图中定义参数属性，也可以在基于图标的树视图中定义参数属性。

1. 使用参数列表

使用参数列表可以在任意时刻查看所有的参数、创建新的参数、删除参数，或者修改已经存在参数的属性。

① 单击参数列表按钮或者选择“Vuser”>“Parameter List”，打开参数列表对话框，如图 5-37 所示。

② 要创建新的参数，单击“New”按钮，新的参数则被添加在参数树中，该参数有一个临时的名字，你可以给它重新命名，然后回车。设置参数的类型和属

性，单击“OK”按钮，关闭参数列表对话框。

注意　不要将一个参数命名为“unique”，因为这个名称是用户脚本生成器本身的。用户脚本生成器创建新的参数，但是不会自动用该参数在脚本中替换任意选中的字符串。

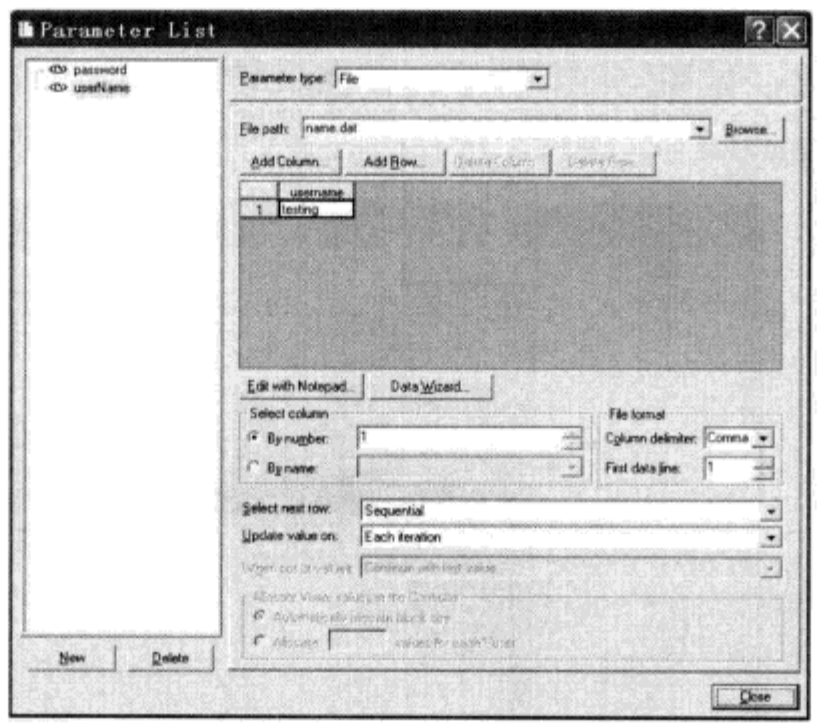


图 5-37　VU 参数列表对话框

③ 要删除已有的参数，首先要从参数树中选择该参数，单击“Delete”按钮，然后确认你的行为即可。

④ 要修改已有的参数，首先要从参数树中选择该参数，然后编辑参数的类型和属性。

2. 数据文件

数据文件包含着脚本执行过程中虚拟用户访问的数据。局部和全局文件中都可以存储数据。可以指定现有的 ASCII 文件、用脚本生成器创建一个新的文件或者引入一个数据库。数据文件中的数据是以表的形式存储的。一个文件中可以包含很多参数值。每一列包含一个参数的数据，列之间用分隔符隔开，比如用逗号。

如果使用文件作为参数的数据源，必须指定以下内容：文件的名称和位置、包含数据的列、文件格式、包括列的分隔符、更新方法。

如果参数的类型是“File”，打开参数属性（Parameter Properties）对话框，设置文件属性如下：

① 在“File path”中输入文件的位置，或者单击“Browse”按钮指定一个已有文件的位置，如图 5-38 所示。在默认情况下，所有新的数据文件名都是“parameter_name.dat”，注意，已有的数据文件的后缀必须是.dat。

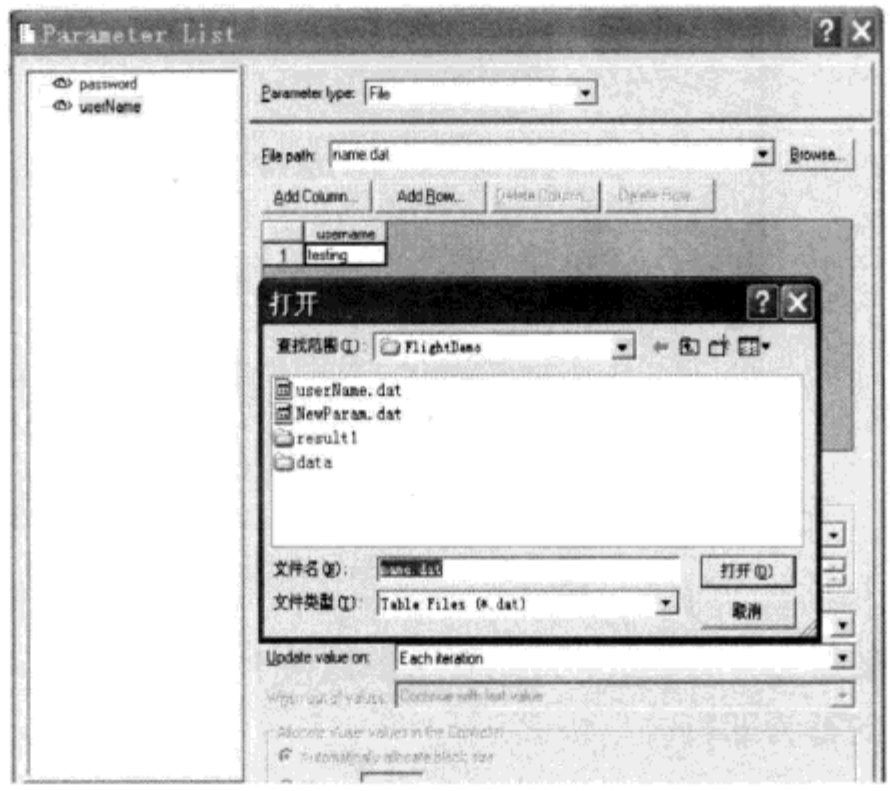


图 5-38 选择打开文件来导入参数

② 单击“Edit With Notepad”按钮，打开记事本，里面第一行是参数的名称，第二行是参数的初始值。使用诸如逗号之类的分隔符将列隔开。对于每一个新的表行开始一行新的数据。

注 意 在没有启动记事本的情况下如果想添加列，就在参数属性对话框中单击“Add Column”按钮，打开“Add new column”对话框。输入新列的名称，单击“OK”按钮，脚本生成器就会将该列添加到表中，并显示该列的初始值。

③ 在“Select column”部分，指明选择参数数据的列。可以指定列名或者列号。列号是包含你所需要数据的列的索引；列名显示在每列的第一行（row 0）。

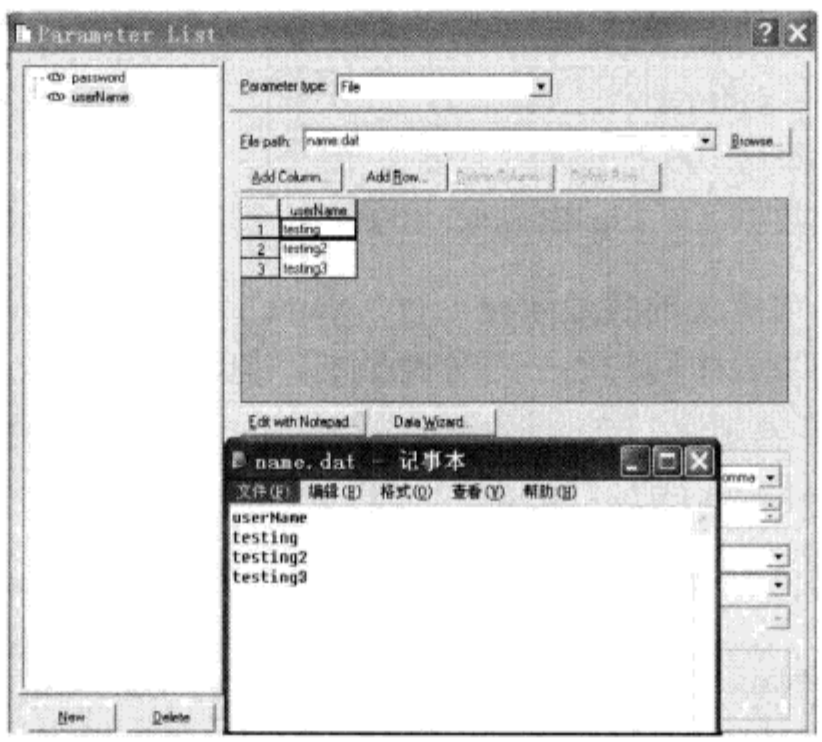


图 5-39　以记事本方式打开参数

④ 在“Column delimiter”中输入列分隔符，可以指定逗号、空格符等。

⑤ 在“First data line”中，在脚本执行的时候选择第一行数据使用。列标题是第 0 行，若从列标题后面的第一行开始的话，那么就在“First data line”中输入 1；如果没有列标题，就输入 0。

⑥ 在“Select next row”中输入更新方法，以说明虚拟用户在获取第一行数据后，下一行数据按照什么规则来取。方法可以是：顺序的（Sequential）、随机的（Random）、唯一的（Unique），或者与其他参数表相同的行（Same Line as..）。下面我们一一介绍。

（1）顺序（Sequential）：该方法顺序地给虚拟用户分配参数值。如果正在运行的虚拟用户访问数据表的时候，它会取到下一行中可用的数据。也就是说，按着顺序一个个取，取了第一行再取第二行，取了第二行再取第三行，依此类推。如果参数表里的数据都取一遍了，不要紧，那就再回到第一行，重新开始。

（2）随机（Random）：该方法在每次迭代的时候会从数据表中随机取一行的数据。比如当前参数表中有 100 行数据，那么随机数就从 1~100 之间任取一个，然后作为行号，去取相应行的参数值数据。

（3）唯一（Unique）：Unique 方法分配一个唯一的有顺序的值给每个虚拟用户的参数。也就是 100 行数据，只能取 100 次，如果第 101 个用户来取，怎么办？对不起，没有数据了，LoadRunner 会报错，提示数据不够用。

（4）与以前定义的参数取同一行（Same Line As <parameter>）：该方法从与以前定义过的参数中同样的一行分配数据，但必须指定包含有该数据的列。在下拉列表中会出现定义过的所有参数列表。注意：至少其中的一个参数必须是 Sequential、Random 或者 Unique。

如果数据表中有 3 列，3 个参数定义在列表中：id1、name1 和 title1，如下：

ID	Name	Title
132	Kim	Manager
187	Cassie	Engineer
189	Jane	VP

对于参数 id1，你可以指示虚拟用户使用 Random 方法，而为参数 name1 和 title1 就可以指定方法“Same Line as id1”。所以，一旦 ID“132”被使用，那么，姓名（Name）“Kim”和职位（Title）“Manager”就同时被使用。

⑦ Updtae value on，数据的更新方法。

我们做事主要考虑 3 个因素：什么时候做，什么地点做，还有怎样做。对应参数表的读取规则来说，上面的 Select next row 指的是怎么取新值，是顺序还是随机等。而这里 Update value on 指的是什么时候取新值。

注 意 LoadRunner 并不是每次取值都是要取新值的。所以就有了以下几种取新值的策略：

- Each iteration——每次迭代就要取新值（在同一个迭代中，无论读几次参数，获得的都是同一个参数值）。

- Each occurrence——只要取一次，就要新的（在同一个迭代中，读一次参数，就要取其新值，而新值是从哪里来的，由 Select next row 来规定）。
- Once——在所有的循环中都使用同一个值（只取一次，也就是说，这个参数只有一个值）。

⑧ When out of values，超出范围：（选择数据为 Unique 时才可用到）

- Abort Vuser——中止。
- Continue in a cyclic manner——继续循环取值。
- Continue with last value——取最后一个值。

⑨ Allocate Vuser values in the Controller 在控制器中分配值：（选择数据为 unique 时才可用到）

- Automatically allocate block size——自动分配。
- Allocate()values for each Vuser——指定一个值。

实例

场景需求：100 个不同的用户以各自的密码并发登录某搜索引擎系统，然后每个用户做 10 个不同关键字的搜索，最后退出搜索系统。问参数表该如何设计？

解析

很显然，在此场景下，我们至少需要 3 个参数：username、password 和 keyword，分别存储用户名、密码和关键词。其中 username 参数包含 100 条记录，password 参数包含 100 条记录，keyword 参数包含 100*10=1000 条记录。

对于脚本结构设计，我们从前面章节介绍的内容中已经知道，可以把登录的操作放在 vuser_init 中，搜索操作放在 Action 中，迭代设为 10 次，退出操作放在 vuser_end 中。

我们在参数表中做如下设置。

username：

- Select next row 设为 Unique（或 Sequential）；
- Update value on 设为 Each iteration。

password：

- Select next row 设为 Same Line as username（为了保证 username 和 password 一一对应）；
- Update value on 设置自动与 username 相同。

keyword：

- Select next row 设为 Unique（或 Sequential）；
- Update value on 设为 Each iteration。

? 思考

如果我们更改上面的脚本结构设计，取消迭代，而把 10 次搜索操作放在同一个 Action 中。

参数表的解决方案是：

username 和 password 与第一种情形保持不变。

keyword：

- Select next row 设为 Unique（或 Sequential）；
- Update value on 设为 Each occurrence。

思考一下这样修改的原因何在。

5.7.4 高级——从已存在的数据库中导入参数数据

LoadRunner 允许你利用参数化从已经存在的数据库中导入数据。可以使用下列两种方式之一：

（1）使用 Microsoft Query（要求在系统上先安装 MS Query）。

（2）指定数据库连接字符串和 SQL 语句。

用户脚本生成器在从数据库中导入数据的过程中提供了一个向导。在向导中，指明如何导入数据——通过 MS Query 创建查询语句或者直接书写 SQL 语句。在导入数据以后，以.dat 为后缀并作为正规的参数文件保存。要开始导入数据库中数据的过程，在参数属性对话框中单击“Data Wizard”按钮，则打开数据库查询向导。

1．创建新的查询

① 选择“Create query using Microsoft Query”。如果需要 MS Query 的帮助，选择“Show me how to use Microsoft Query”，然后单击“Finish”按钮，如图 5-40 所示。

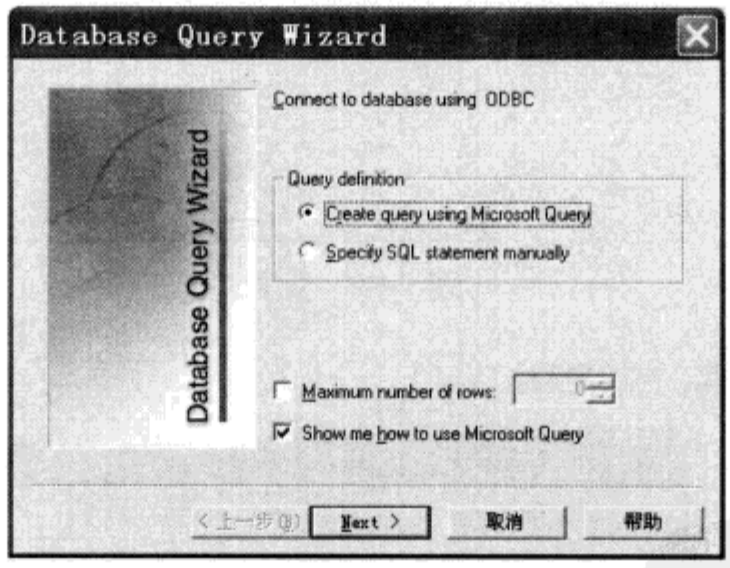


图 5-40　在 Data Wizard 中选择通过 Microsoft Query 创建查询

如果你还没有安装 Microsoft Query，LoadRunner 会提示你这个功能不可用。在进行操作之前，从 Microsoft Office 中安装 MS Query。

② 在 Microsoft Query 中导入期望的表和列。

- 选择表和字段（见图 5-41）



图 5-41 选择表和字段

- 设置过滤条件（见图 5-42）

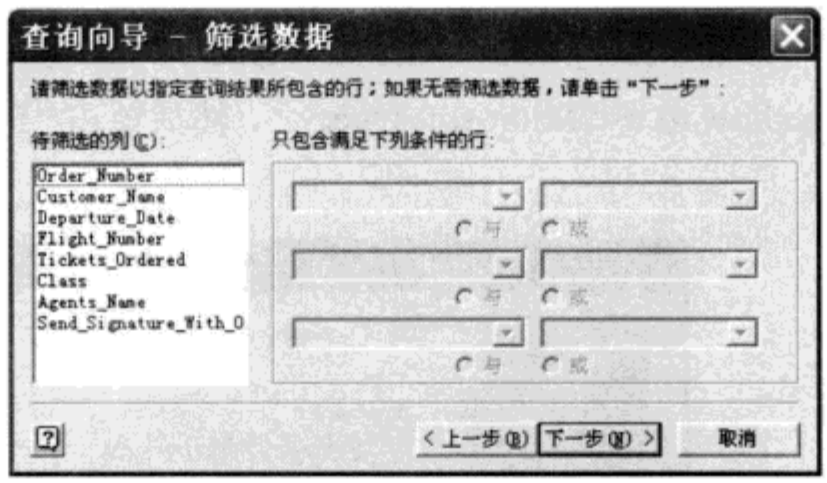


图 5-42 设置过滤条件

- 设置排序顺序（见图 5-43）

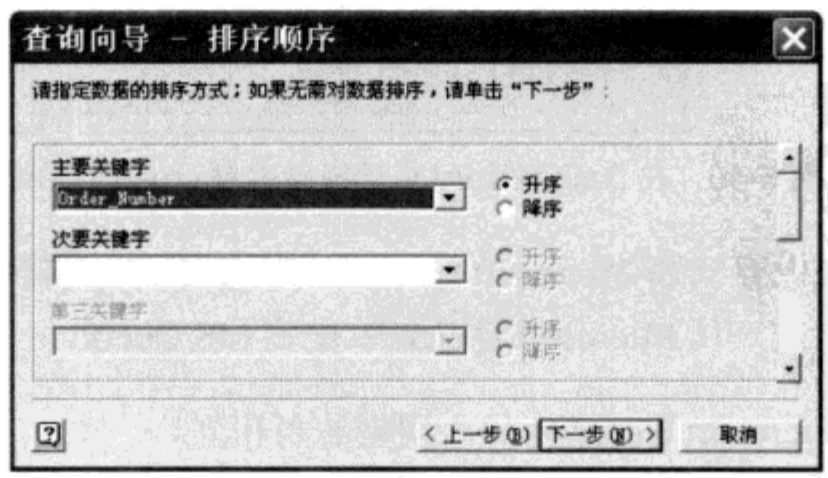


图 5-43 设置排序顺序

③ 在完成数据的导入后，选择“Exit and return to Mercury Virtual User Generator”，然后单击“完成”按钮，如图 5-44 所示。

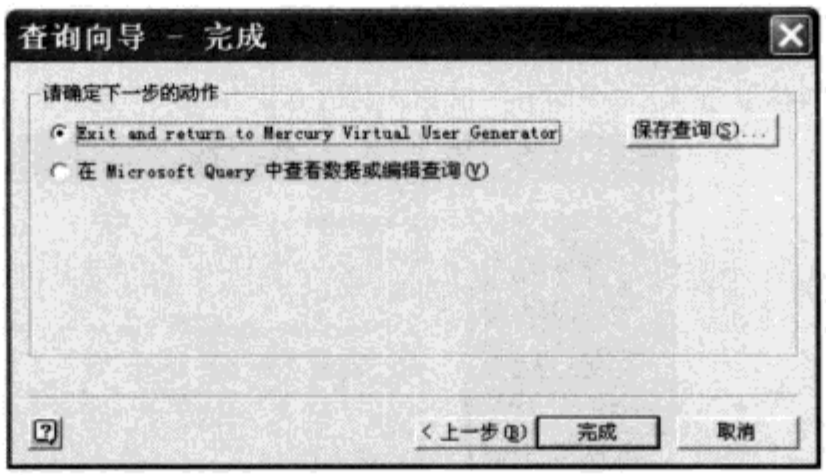


图 5-44　完成查询并选择返回

在参数列表对话框中，数据库记录以 data 文件的形式显示出来，如图 5-45 所示。

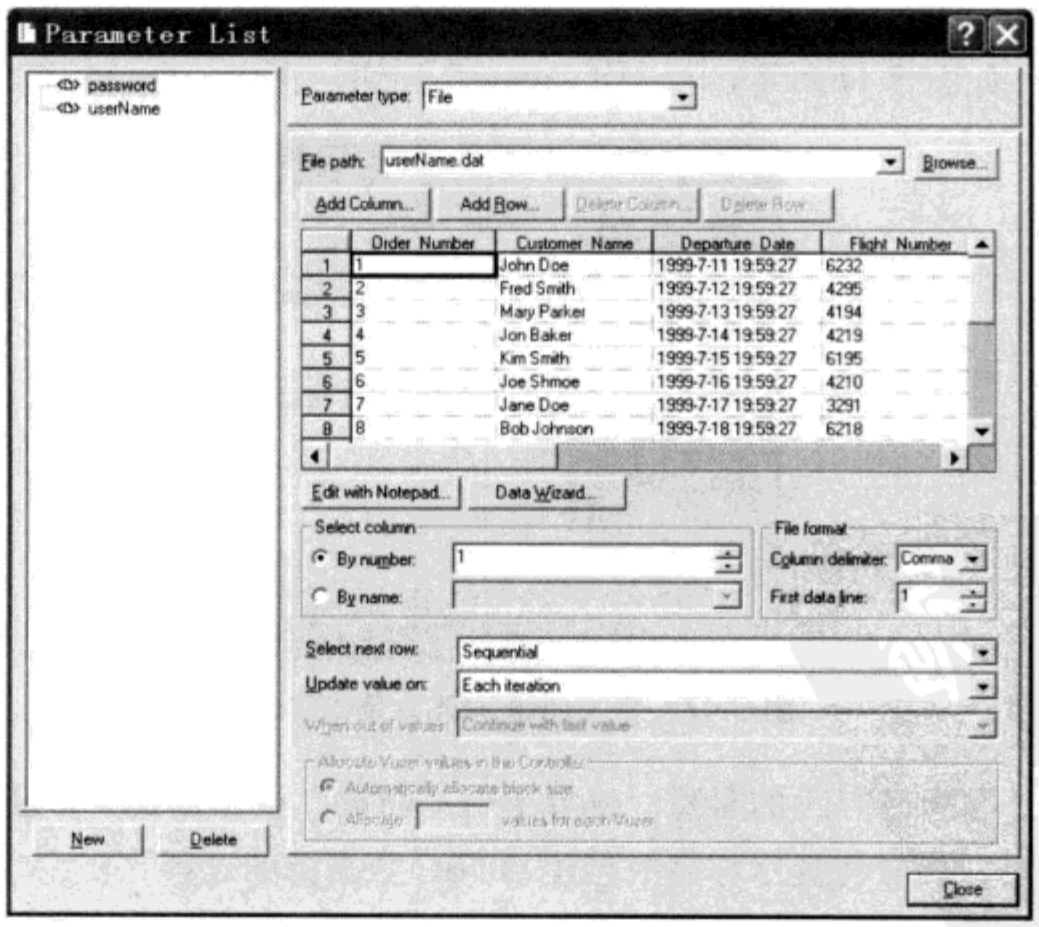


图 5-45　参数列表中返回所查询的数据

2. 指定数据库连接或者 SQL 语句

① 如图 5-46 所示，选择“Specify SQL statement manually”，然后单击“下一步”按钮。

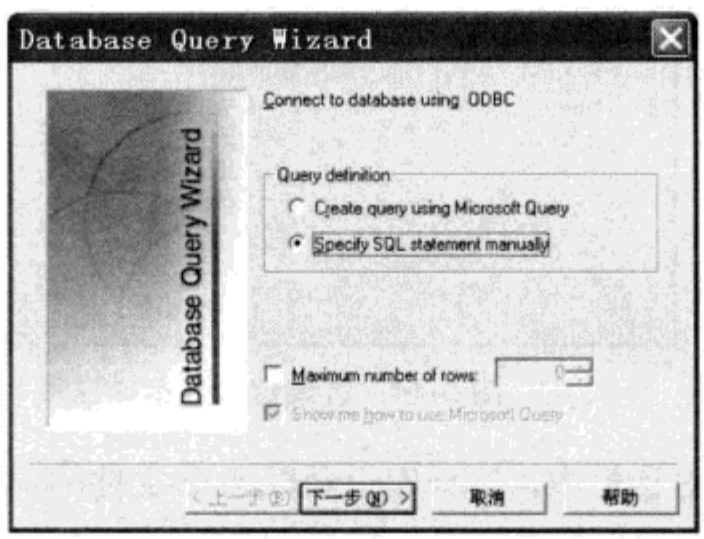


图 5-46 在 Data Wizard 中选定手工写入 SQL 语句

② 如图 5-47 所示，单击“Create”按钮，指定一个新的连接字符串。

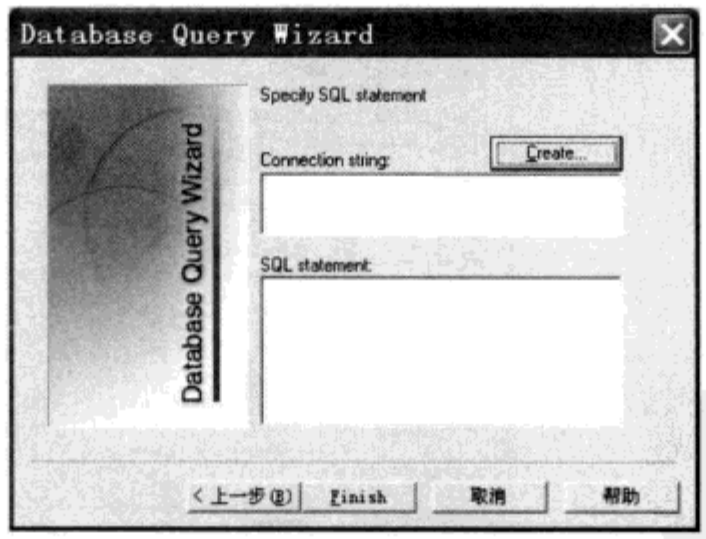


图 5-47 手工创建 SQL，单击“Create”按钮

③ 选择已有的数据源，或者单击“新建”按钮创建一个新的数据源，如图 5-48 所示。向导将提示你完成创建 ODBC 数据源的过程。在完成后，连接字符串就会在连接字符串框中显示出来。

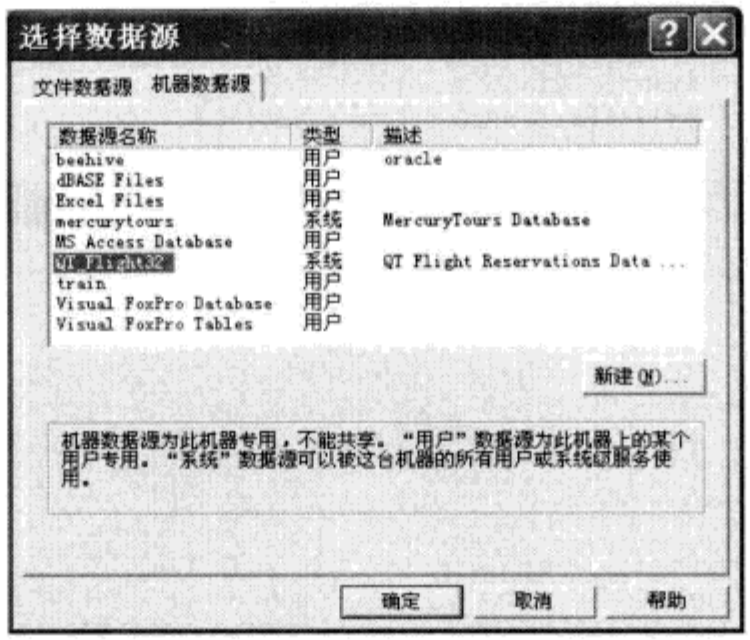


图 5-48　在 ODBC 数据源中选择相应的数据源

④ 在 SQL statement 框中，输入或者粘贴 SQL 语句，如图 5-49 所示。

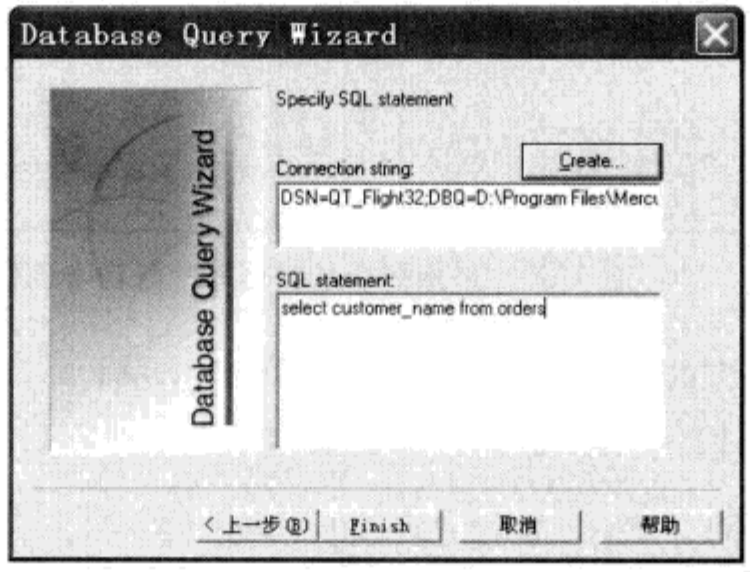


图 5-49　数据库连接串被创建

⑤ 单击“Finish”按钮继续 SQL 语句并导入数据。数据库记录将以 data 文件的形式显示在参数列表对话框中，如图 5-50 所示。

⑥ 在“Select column”部分中，指定包含当前参数数据的列。你可以指定列号或者列名。

⑦ 从“Select next row”列表中选择一个更新方法来告诉虚拟用户在脚本指

定的过程中如何选择表中的数据。可选项是：Sequential、Random、Unique 或者 Same Line as。

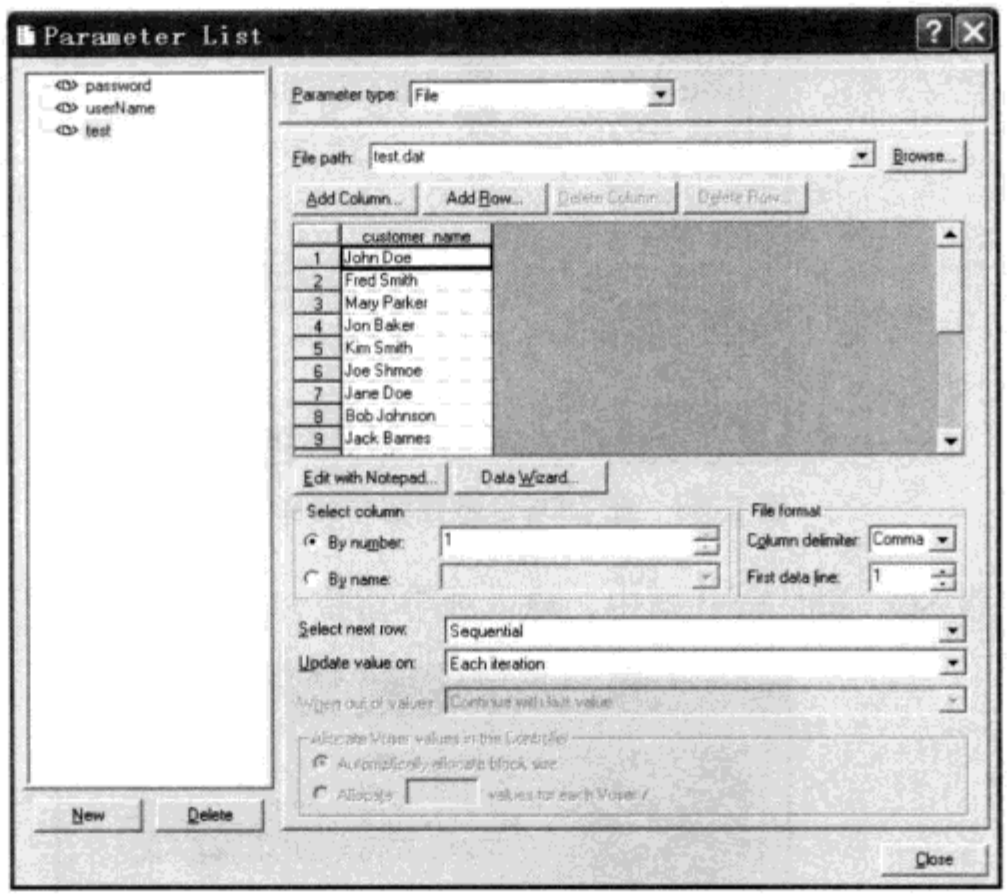


图 5-50 参数列表从数据库中成功得到数据

⑧ 如果从 Update value on 中选择“Each iteration”，虚拟用户在每次迭代的时候都会使用新的一行数据而不是重复同样的数据。

5.8 检查点（Check point）

5.8.1 为什么需要检查点

LoadRunner 的很多 API 函数的返回值会改变脚本的运行结果。比如 web_find 函数，如果它查找匹配的结果为空，它的返回值就是 LR_FAIL，整个脚本的运行结果也将置为 FAIL；反之，查找匹配成功，则 web_find 返回值是 LR_PASS，整个脚本的运行结果置为 PASS。而脚本的结果则反应在 Controller 的状态面板上和

Analysis 统计结果中。



在 VU 函数手册中，点击函数的 Return Value 项，可查看此函数是否返回 LR_PASS/LR_FAIL，如图 5-51 所示。

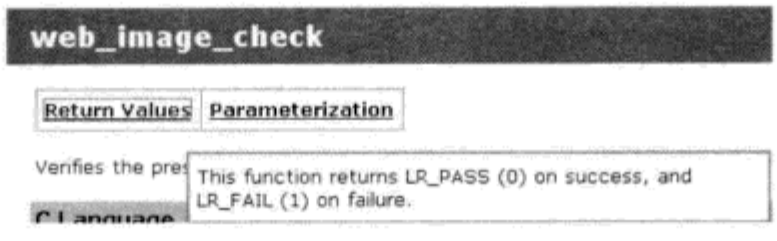


图 5-51　在 function help 中查看函数的返回值

上图说明 web_image_check 也是一个决定脚本运行结果的函数。

但仅仅通过脚本函数执行结果来决定整个脚本的成功/失败，这未免太草率了。因为脚本往往是在执行一个业务流程，VU 脚本函数本身是协议级的，它执行的失败会引起整个业务的失败，但它运行成功却未必意味着业务会成功。比如，我们要测 100 人登录一个 Web 邮件系统，此邮件系统有个限制，即不允许使用同一个 IP 登录两个用户。显然，如果 LoadRunner 没有开启多 IP 欺骗功能的话，第一个虚拟用户登录成功后，第二个虚拟用户试图登录，系统将返回一个页面，提示用户“您已经登录本系统，请不要重复登录！”。在这种场景下，如果没有设检查点来判断这个页面，那么 VU 认为它已经成功地发送了请求，并接到了页面结果（http 状态码为 200，虽然是个错误页面）。这样 VU 就认为这个动作是成功的。但事实如我们所见，并非如此。因此我们要采用检查点来判断结果。

检查点（Check Point）并不是一个 LoadRunner 里专有的概念。在 WinRunner 和 QTP 中就有检查点。对于自动化测试来讲，检查点是一个很重要的功能，它的作用是验证程序的运行结果是否与预期结果相符。

对于 Web Vuser 类型，有两种设置检查点方法，下面进行介绍。

5.8.2 检查点实施之一：ContentCheck 定义

在【Run-time settings】>【ContentCheck】中（如图 5-52 所示），这里的设置是为了让 Vugen 检测何种页面为错误页面。如果被测的 Web 应用没有使用自定义的错误页面，那么这里不用作更改；如果被测的 Web 应用使用了自定义的错误页面，那么这里需要定义，以便让 Vugen 在运行过程中检测，服务器返回的页面是否包含预定义的字符串，进而判断该页面是否为错误页面。如果是，Vugen 就停止运行，指示运行失败。

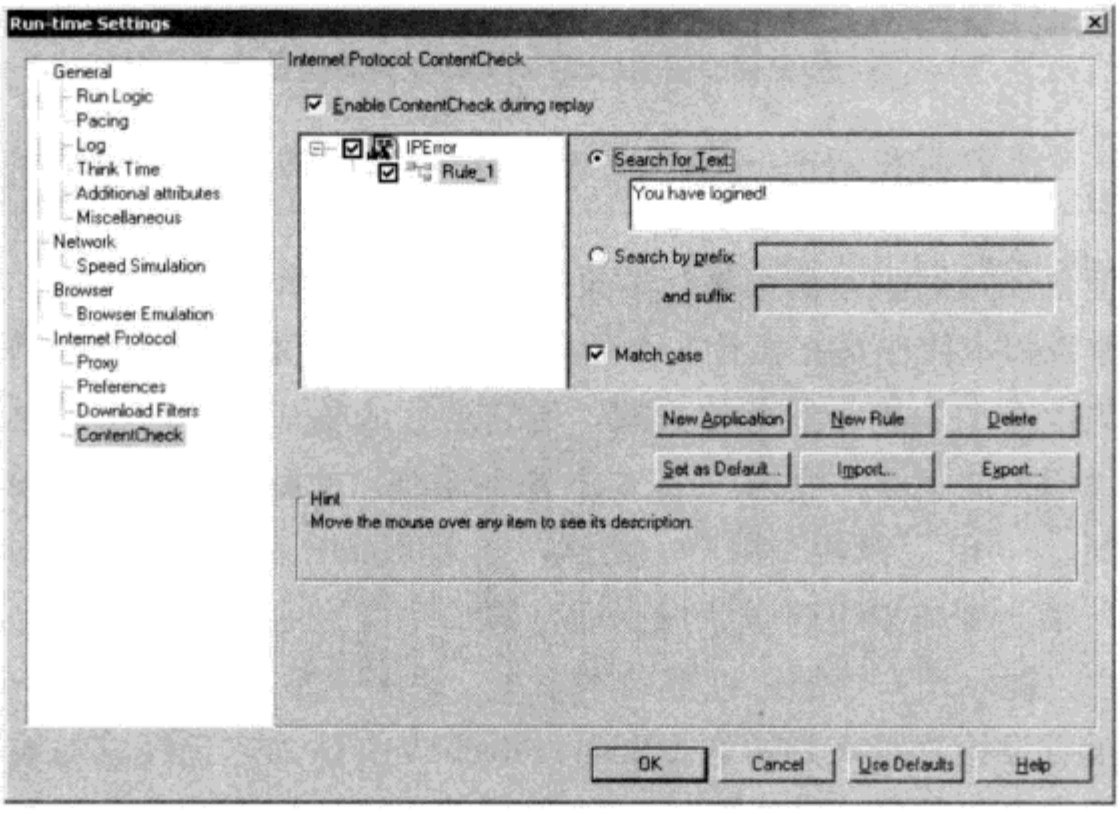


图 5-52 通过 ContentCheck 定义检查点

5.8.3 检查点实施之二：检查函数

另外一种设置方法是在 Web Vuser 里，通过检查函数来完成检查点功能。Web Vuser 提供 Image Check 和 Text Check 两种方式。其原理就是在上一个请求页面的函数完成后，运行检查函数，在结果页面中搜索既定的图片/关键文字。

以下是取自 LoadRunner 函数手册的一个 Text Check 例子：

```
    Web_url("index.html",
        "URL=http://server1/people/employees.html",
        "TargetFrame=",
        LAST);
    web_find("Employee Check",
        "expect=notfound",
        "matchcase=yes",
        "onfailure=abort",
        "report=failure",
        "repeat=no",
        "what=John",
        LAST);
    }
```

在这个例子中，web_find 函数在 employees.html 中搜索“john”关键字。有关 web_find 函数的各个参数的含义以及使用方法，可参看 LoadRunner 随带的函数手册。

Image Check 的功能则由另外一个函数 web_image_check 实现：

```
web.url("index.html",
        "URL=http://localhost/ImagesAndMaps.html",
        new String [] {
            "TargetFrame=", "LAST"});
web_image_check("Go2Venus",
            new String []{"Alt=Venus", web.LAST});
```

这同样是一个 Web Vuser 脚本，是用 Java 实现的，而不是录制时默认生成的 C 语言。Web_image_check 则在 ImageAndMaps.html 中查找 alt 属性为“venus”的图片。



提示 LoadRunner 一直是使用 C 作为脚本语言的。在 Java 普及应用的当今，LoadRunner 开始加大对 Java 的支持力度，这表现在 LoadRunner 对原先运行在 C 基础上的 Vuser 同样提供了 Java 运行环境，而一些经常使用的 LoadRunner C 函数也有了相同的 Java 函数实现。比如在 Web Vuser 中，原先的 web_url 函数摇身一变，在 Java 中成了 web.url；另外，对于 CORBA、RMI 的 Vuser，VU 可

以直接录制生成 Java 脚本。需要注意的是：Java 脚本的录制选项和关联方法等与 C 脚本都有一些差别，可以参看 LoadRunner 的 VU 手册。

如果我们看过 LoadRunner 的函数手册，就会发现 VU 的 Web Vuser 还提供了和 web_find 十分貌似的另外一个检查点函数：web_reg_find。

web_reg_find 里的 reg 意为注册（register）。因此 web_reg_find 和 web_find 的不同之处是 web_reg_find 是先注册，后查找；而 web_find 是查找前面的请求结果。因此，我们在使用 web_reg_find 函数的时候，将它放在请求语句的前面，如下：

```
web_reg_find("Text=ABC", "SaveCount=abc_count", LAST);
web_url("MercuryWebTours",
        "URL=http://localhost/hello.html",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);
if (strcmp(lr_eval_string("{abc_count}"), "0") == 0)
Action A
else
Action B
```

上面的脚本运行过程是：如果 web_reg_find 在 hello.html 页面中没有找到“welcome”字符串，则执行 Action A；如果找到了一次或一次以上，则执行 Action B。

我们可以看到，web_find 和 web_reg_find 函数两者是有一些差别的：

（1）web_reg_find 先注册的优势是脚本能够一边接收 Server 的数据缓冲，一边进行查找，提高了查找的效率。

（2）web_reg_find 的参数与 web_find 并不完全一样，其中有个参数叫做 SaveCount，它能够记录查找匹配的次数。而 web_find 的机制是一旦查找匹配成功，

就立即返回，并不继续查找和记录匹配次数。

（3）VU run time 设置中的“enable image and text check”对 web_find 有效，而对 web_reg_find 无效。

5.8.4　检查点设置技巧

如何加入检查点，才能检查出正确的结果。与事实相符，这的确有一些技巧：

（1）它必须满足是验证事务通过与否的充分必要条件。检查点通过，我们就能够确信系统是一个正常的状态。

（2）检查点可以是常量，也可以是变量。

（3）检查点可以是文本、图像文件，也可以是数据库记录等。

5.9　高级——多 Action

对于支持多 Action 的 Vuser，可以把一个脚本的 Action import 到另外一个脚本中。当然必须保证这两个 import 的脚本 Vuser 类型是相同的。注意，被 import 的 Action 中的有关参数也一同被 improt。选择被 import 的脚本，然后选择被 import 的 Action。步骤如下：

① 选择“Action”>“Import Action into vuser”，会显示对话框，提示选择脚本。

② 选择要 import 的 Action，单击确定按钮。

③ 生成一个新的 Action，名为 Imported_Action，函数都被 import 在此中。

在“Run-time Settings”>“Run Logic”中调整 Action 的运行策略。

打开“Run-time Settings”（运行时设置）对话框，然后选择“General”下的“Run logic”节点，如图 5-53 所示。

（1）迭代次数（Number of Iterations）：迭代的次数。LoadRunner 将按指定的次数重复执行所有 Actions，但不会重复 Vuser 脚本的 vuser_init 和 vuser_end 部分。

需要注意的是，如果在 Controller 的计划设置中指定了方案持续时间，则持续时间设置将覆盖 Vuser 迭代设置。这意味着，如果将持续时间设置为 5 分钟（默认设置），Vuser 将在 5 分钟内按照需要继续运行任意多次迭代，即使运行时设置仅指定一次迭代。

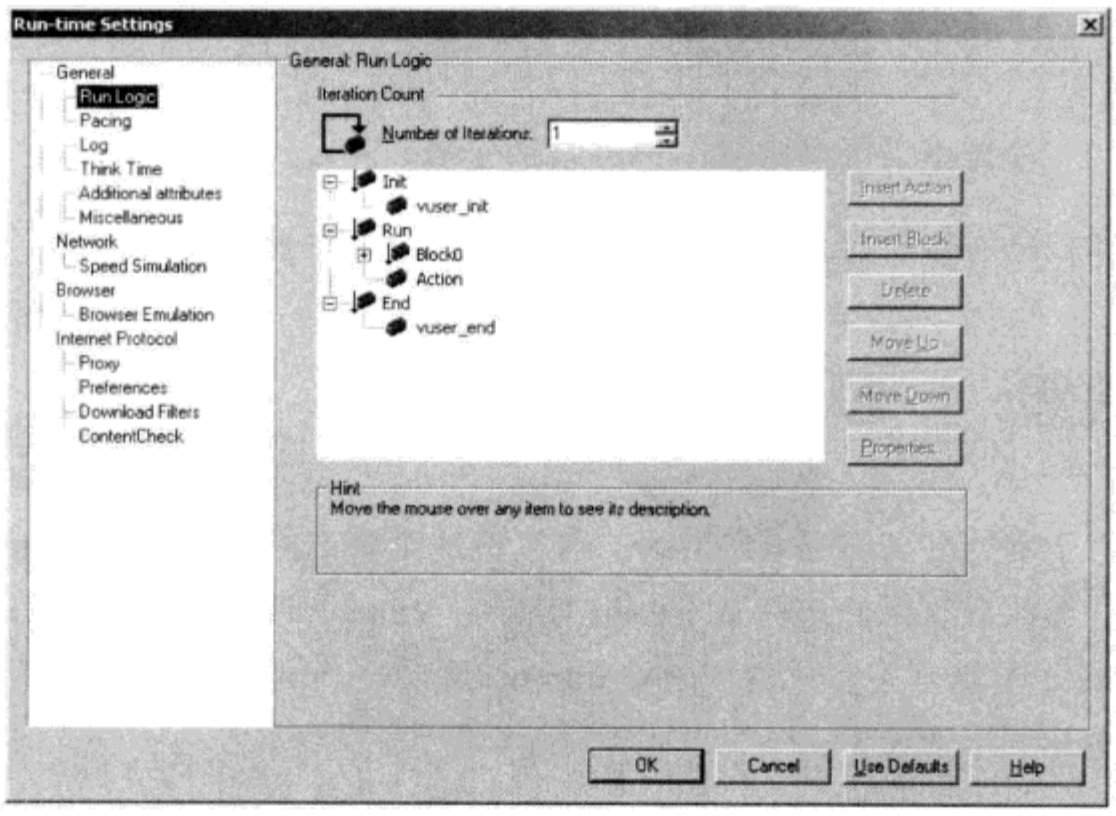


图 5-53　在 Run-time Settings 中设置运行逻辑

（2）操作块（Block）：操作块是脚本内函数的组合。每个块都有自己的单独属性设置，包括顺序、迭代和权重。可以通过单击“Properties”设置 Block 的属性。

（3）顺序（Sequence）：可以设置块内脚本的操作顺序。有两种方式：按顺序和随机执行操作。

（4）迭代（Iteration）：除了为整个 Run 部分设置迭代次数以外，还可以设

置单个操作或操作块的迭代。该设置非常有用，例如，可以用来模拟一个商业站点，你在该站点执行了多个查询查找某个产品，但只进行了一次采购。



操作块（Block）是 Vuser 脚本的操作组。这是 VU 为用户提供的一个很贴心的功能。

比如在一个银行转账系统中，系统提供多个业务，有查询、转账、储蓄等业务，但每个业务被执行的概率可能会不一样。在这种情况下，我们可以把查询设为 Block1，转账设为 Blcok2，储蓄设为 Block3，各个 Block 的 Iteration 属性体现了它们的概率分配，同时“登录”和“注销”操作是这三个 Block 所共有的。这样，执行一次脚本，就执行了所有的业务。当然，如果 VU 没有 Block 功能，我们也可以使用脚本 C 语言的循环机制来实现。有兴趣的朋友可以思考一下如何做。

④ 运行脚本。验证 Action 确实按照我们在 Run-time Settings 中的设置运行。



运行时设置（Run-time Settings）

VU 中有运行时设置（Run-time Settings），而 Controller 在运行时也同样有一个运行时设置。这两个运行时设置虽然看起来是一样的，但其实并不出自一处，而是分别存储在各自的环境下。VU 的 Run-time Settings 存储在 default.cfg 文件中，而 Controller 则把自己的 Run-time Settings 放在自己的场景文件下。

这也就是说，当通过脚本创建场景后，再修改脚本的 Run-time Settings，这不会影响到 Controller 的 Run-time Settings。

而在 LoadRunner 早期版本中（6.5 以前），VU 和 Controller 是共享同一个 Run-time Settings 的。后来把两者分开，使它们“各自为政”，应该是出自方便的考虑。比如，在 VU 中我们修改 Run-time Settings 很多是为了 debug 脚本，输出尽可能多的日志，忽略 Think Time；而 Controller 中运行脚本则是开始性能测试，这时的 Run-time Settings 策略会有变化。

5.10 高级——脚本错误处理机制

一个好的脚本应该具有健壮性，要能够捕捉到错误，并能采取有效的错误处理方式，否则脚本一旦发生错误，唯一的出路就是退出执行。

VU 提供了一套在出错情况下的脚本处理机制。VU 的错误处理机制可设定 Vuser 在执行脚本时遇到错误怎样处理。当 Vuser 遇到错误时，可以有两种处理策略：一是停止执行脚本，这适用于严重的问题；二是忽略这个错误，继续执行下去。

在默认情况下，当检测到错误时，Vuser 将停止执行脚本，在 Run-time Settings 中我们就可以看到它的默认设置，如图 5-54 所示。

如不想 VU 在出错后结束，而是继续执行。那么在运行时设置中，可以勾选“Continue on error”（出现错误时仍继续）。这个设置适用于整个 Vuser 脚本，它是一个全局开关。

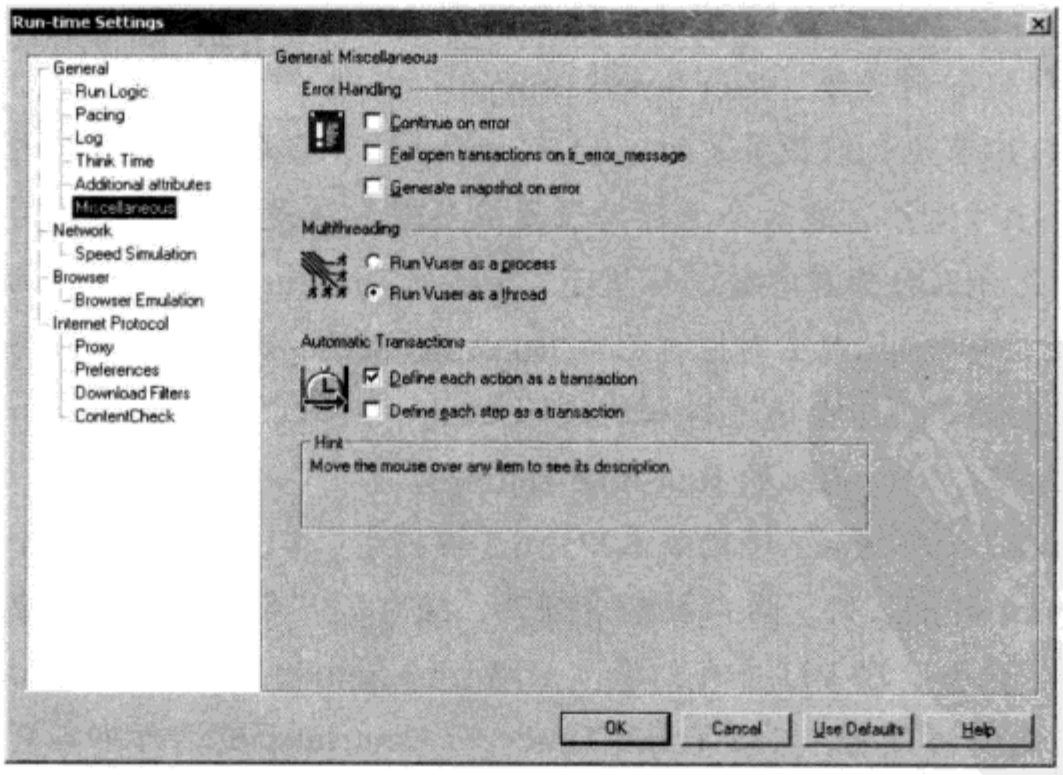


图 5-54 在 Run-time Settings 中设置出错处理方式

但这种方法有一个毛病，就是不够灵活，有“一竿子打翻一船人”之嫌，我们更希望脚本能够“具体情况具体分析”，能够对于不同的函数应用不同的错误处理机制。既然信不过 LoadRunner，那么解决的办法就是我们自己来干，在脚本中加入 lr_continue_error 函数。

使用 lr_continue_on_error 函数。通过 lr_continue_on_error 函数可以控制 Vuser 脚本特定段的错误处理。要标记该段，就用 lr_continue_on_error(1)和 lr_continue_on_error(0)语句将其括起来。

使用 lr_continue_on_error 函数的脚本段将会覆盖“出现错误时仍继续”的 Run-time Settings 运行时设置。

例如，下面的脚本，其运行时设置没有勾选“Continue on error”，VU 如果访问 www.51testing.com 遇到错误，将会结束执行脚本。

```
Web_lin("test",
    "text = www.51testing.com",
    LAST);
Lr_output_message("finished linking");
```

运行时，如 web_link 出错，“finished linking”这段文字永远也不会输出。但是如果我们认为这个 web_link 函数无关紧要，它即使失败，我们也要继续执行脚本。那么解决的办法是用相应的 lr_continue_on_error 语句将该段括起来：

```
lr_continue_on_error(1);
Web_lin("test",
    "text = www.51testing.com",
    LAST);
Lr_continue_on_error(0);
Lr_output_message(" finished linking");
```

在上面的脚本中，即使 web_link 函数执行失败，lr_output_message("finished linking")也会被继续执行。第一个 lr_continue_error(1)是将“Continue on error”设为 true，其后面的语句的错误处理机制都被应用为“Continue on error”，直到 VU 遇见 lr_continue_error(0)，再将“Continue on error”恢复成 false。

5.11 高级——脚本调试技巧

解释性语言的调试一直是一个棘手的问题。如果没有专门的 debug 工具，只能通过加入变量输出语句来查看每个变量值的变化。比如，在过去 JavaScript 调试的常用方法就是删除所有代码，然后一行行地增加直到错误出现，这是相当麻烦的。VU 的 C 脚本虽然也像 VC 一样提供了断点、单步运行等功能，但并不能像 VC 一样在调试状态下查看变量、表达式和内存值。因此，个人认为 VU 的调试功能还有待增强。

下面是两种比较常用的调试脚本的方式，它们并不是真正意义上的调试，只能算作“土办法”，但比较有效。

5.11.1 动态运行（Animated run）与非动态运行（Non-Animated run）

脚本可以设置成为 Animated 运行模式和非 Animated 运行模式。在 Animated 运行模式下，VU 会显亮当前正在执行的脚本语句。用户可以通过 VU 设置语句执行的延迟时间，以便观察每行代码执行的效果；相比之下，在非 Animated 运行模式下，VU 在执行脚本时不会显亮语句，而一气执行下去。显然，Animated 运行模式是我们调试脚本的一个很好的方法。

Animated 的设置在 VU“Tools”菜单下的“Options”中，选中“General Options”中的“Replay”选项卡，在此对话框中设置 Animated 的 Replay 时间，生效范围，如图 5-55 所示。

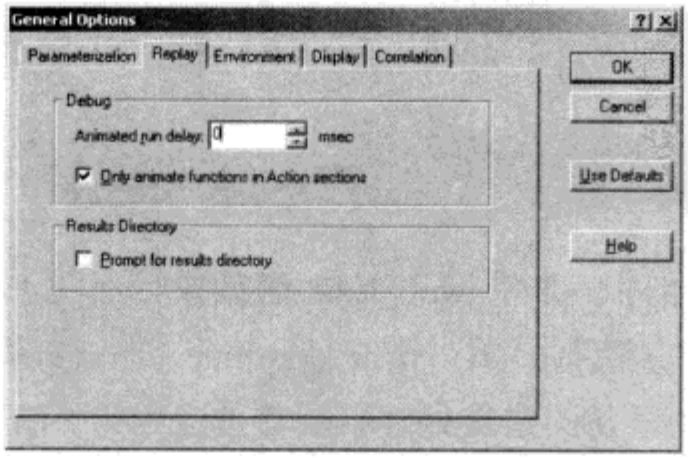


图 5-55 在 General Options 中设定动态运行和非动态运行

然后在“View”菜单中勾选“Animated Run”方式，如图 5-56 所示，则脚本就能以 Animated 的模式运行了。

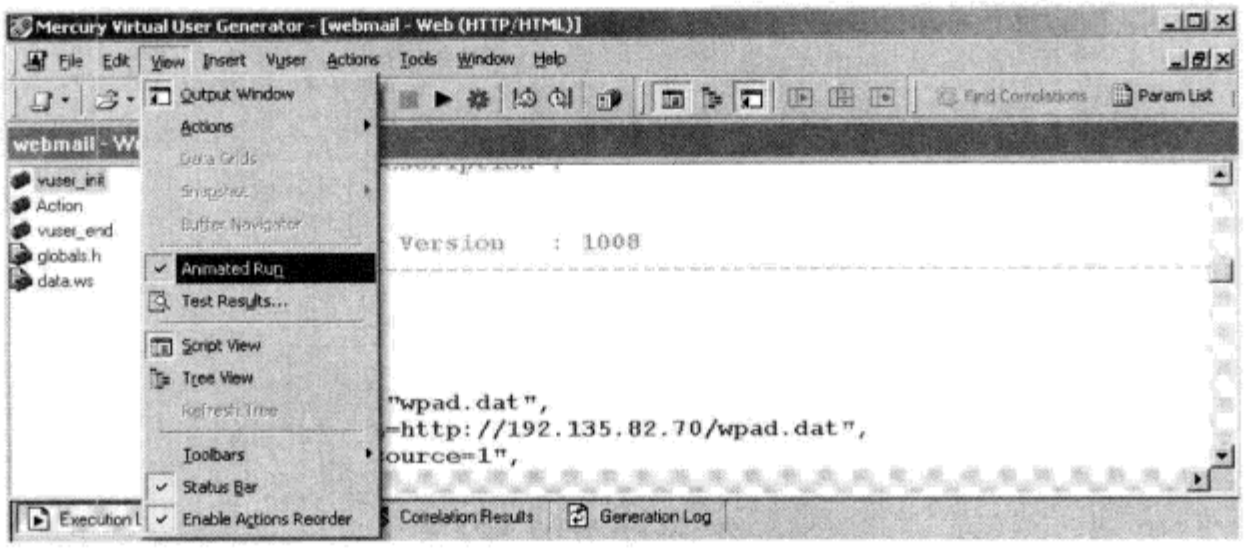


图 5-56　在“View”菜单中勾选动态运行



提示　当执行脚本期间在输出窗口中显示错误消息时，我们可以双击该错误消息，Vugen 将使光标跳到导致问题的测试行；如要得到更详细的出错信息，也可以将光标置于错误代码上并按 F1 键，查看该错误代码的联机帮助解释。

5.11.2　日志设置

对于软件开发人员来讲，日志无疑是跟踪和调试最好的线索。一般在日志中存放的是程序产生的各种信息，包括业务执行、警告和错误。一个优秀的软件应该有一套完备的日志系统，包括日志的内容、格式和级别等内容。在 Java 中，就有专门的 log 类来完成这些工作。同样地，VU 也提供了一套日志系统，我们可以通过 Run-time Settings 的 Log 选项来设置日志的级别，如图 5-57 所示。

（1）仅在出错时发送消息（Send messages only when an error occurs）：指示 Vugen 仅当出错时记录 log。

（2）标准日志（Standard log）：创建在脚本执行期间所发送的函数和消息的标准日志，供调试时使用。大型负载测试方案应禁用该选项。

（3）扩展日志（Extended log）：创建扩展日志，包括警告和其他消息。大型负载测试方案禁用该选项。

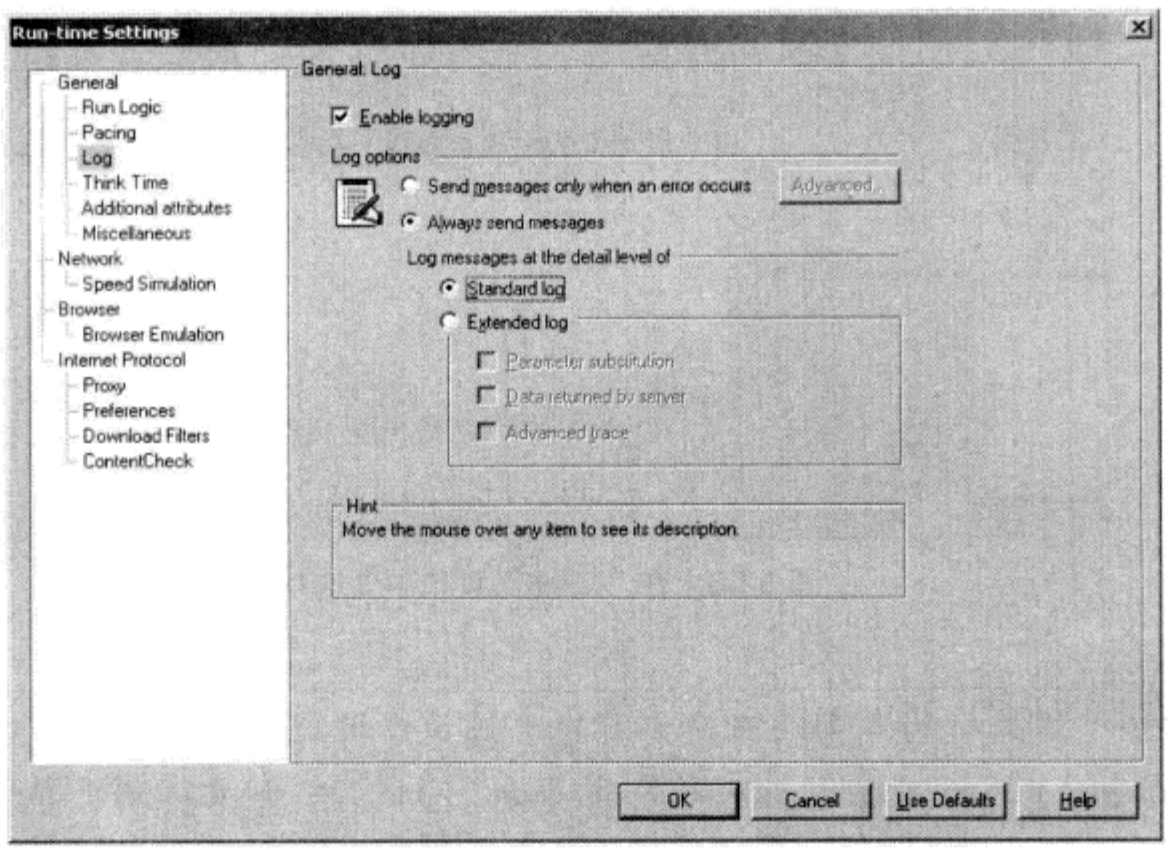


图 5-57　在 Run-time Settings 中设置日志级别

可以使用“扩展日志”选项指定应向扩展日志中添加哪些其他信息：

- 参数替换（Parameter substitution）：选择该选项可记录指定给脚本的所有参数及其相应的值。
- 服务器返回的数据（Data returned by server）：选择该选项可记录服务器返回的所有数据。
- 高级跟踪（Advanced trace）：选择该选项可记录 Vuser 在会话期间发送的所有函数和消息。调试 Vuser 脚本时，该选项非常有用。

但任何事情都有两面性，虽然扩展日志模式很有用，但也会带来麻烦，那就是会产生大量的数据，影响脚本执行速度不说，如果我们想看的信息可能只是某一处，查找起来会像大海捞针一样困难，特别是当面对一个复杂的脚本的时候。

现在，解决问题的办法是选择性地开启扩展日志功能，可以通过 lr_set_debug_message 函数一开一关来关注我们感兴趣的脚本区间，例子如下：

```
void ExtendedLog(int Switch);
ExtendedLog(1) /* 开启扩展日志功能 */
web_link("test",
"Text=test",LAST);
ExtendedLog(0) /* 关闭扩展日志功能*/

void ExtendedLog(int Switch) {
lr_set_debug_message(LR_MSG_CLASS_EXTENDED_LOG,Switch);
lr_set_debug_message(LR_MSG_CLASS_PARAMETERS, Switch);
lr_set_debug_message(LR_MSG_CLASS_RESULT_DATA, Switch);
lr_set_debug_message(LR_MSG_CLASS_FULL_TRACE, Switch);
}
```

如我们在 Run-time Settings 中设置的那样，先是 lr_set_debug_message (LR_MSG_ CLASS_EXTENDED_LOG，Switch)选中了扩展日志模式，然后下面三次调用 lr_set_debug_message 依次选中了“参数替换”、“服务器返回的数据”和“高级跟踪“模式，在 Switch 开关置为 1 的脚本区间内，VU 会记录 web_link 函数执行的完全扩展日志；在 Switch 开关置为 0 后，VU 自动恢复 Run-time Settings 的 Log 设置。这明显的优势是，Run-time Settings 针对整个脚本有效，而 lr_set_debug_message 则可对某段脚本有效，通过控制 Switch 的开关来实现。显然，后者更为灵活。lr_get_debug_message 会得到当前日志的级别，有关 lr_set_debug_message 和 lr_get_debug_message 函数的详细使用方法可参看 LoadRunner 函数手册。



在 Transaction 中加入消息输出语句，可能会增加 Transaction 的执行时间，影响测试的精确程度。



在初次使用 VU 时，VU 编辑器默认的字体不是很好，尤其显示中文的时候。可以通过以下设置来改变 VU 编辑器的字体：在 VU 菜单“Tools”>“General Options”的 Enviroment 选项卡中有一个 Editor 区域，可以在这里更改 VU 的字体。

5.12 高级——编写脚本的最佳实践

有过开发经验的朋友都知道，程序的开发大多不是一蹴而就的，通常要经过代码评审、编码和测试等流程后，才会趋于稳定。而 VU 脚本本身虽然不算复杂，但 VU 脚本运行往往涉及多个因素，也会经常出问题。提早发现和解决脚本中的问题，不仅从技术上下工夫，在流程上也要做文章。通常验证一个脚本的比较好的过程是这样的：

① Generate：录制或开发脚本。

② SUSI（Single User Single Iteration，单用户单循环）：运行录制生成的脚本，解决可能存在的关联问题。

③ SUMI（Single User Multi Iterations，单用户多循环）：参数化脚本，在 Run-time 中设置 Iteration，再次运行，验证参数化问题。

④ MUSI（Multi User Single Iterations，多用户单循环）：Controller 里多用户运行脚本，验证脚本中可能的多线程同步问题。

⑤ MUMI（Multi User Multi Iteration，多用户多循环）：即性能测试开始。

其中①~③是在 VU 中进行的，而④和⑤是在 Controller 中进行的。

VU 编写脚本最佳实践流程图如图 5-58 所示。

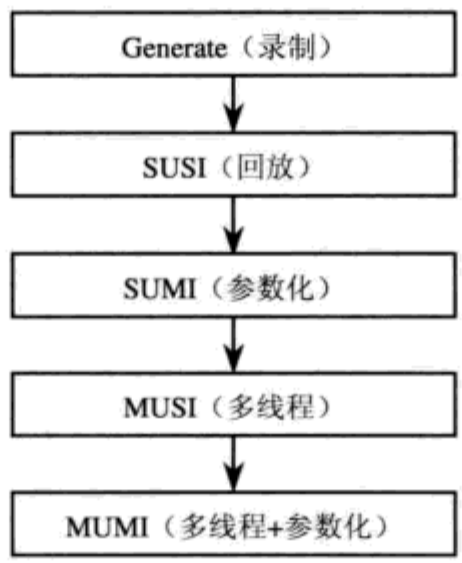


图 5-58 VU 编写脚本最佳实践流程图

作业

使用 VU 录制 http://newtours.demoaut.com/机票预订演示网站，完成一个完整的机票预订流程，包括登录、选择预订条件、下订单。要求如下：

（1）按照最佳实践流程的①~③来完成脚本。

（2）按照关联实践，查看脚本是否有需要关联的数据。

（3）能够在 Controller 中实现 5 个用户并发，选择不同的条件预订机票，考虑 VU 参数表应该怎样设计，怎样修改脚本，并通过设置迭代次数来验证参数表机制。

5.13　VU 脚本运行成功后

脚本运行成功后，就可以创建 Controller 场景并运行场景了。在 VU“Tools”菜单下选择“Create Controller Scenarion”，如图 5-59 所示。

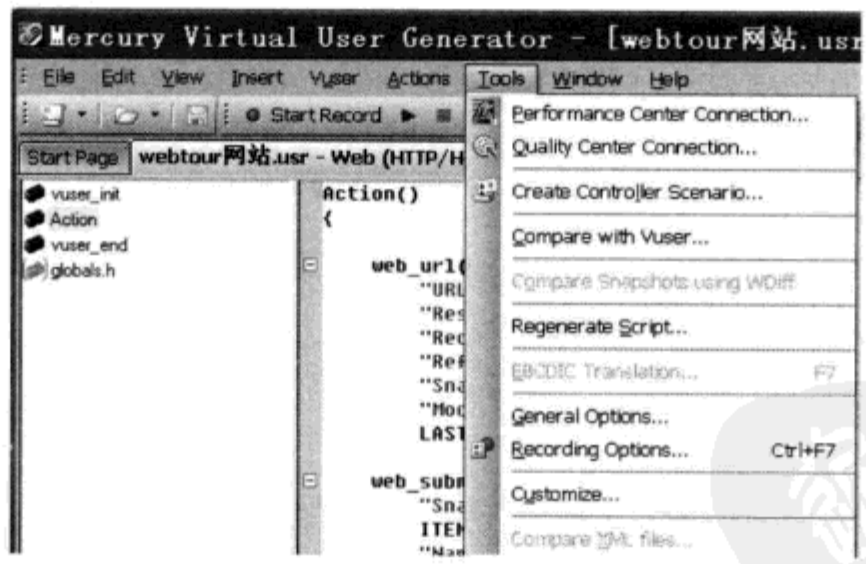


图 5-59　基于 VU 脚本创建 Controller 场景

我们在下一章开始介绍场景控制器（Controller）。



第6章

Controller——性能测试的指挥中心

客户经常会这样描述他们的软件性能需求：

需求一

“每个用户在登录时间小于5秒（包含登录中下载所有资源的时间）情况下的最大用户并发数不能低于500个。”

需求二

“我们的软件系统有2000个用户使用客户端软件进行业务处理，每年通过软件进行处理的总业务量为：150万笔业务/年。”

如果需求一还能让我们看懂的话，那么需求二就是和我们软件人员玩蒙太奇了，光从这句话来看，我们怎么设计性能测试都可能对，但同时，怎么做也都可能是错的。

哲学告诉我们，事物越抽象，外延也就越多，抽象的事物对我们来说也就越“模糊”，“白马非马”就是这个道理；而事物越具体，外延也就越少，但会越明确。层次高的领导不大可能犯错，说错话，底层实施人员经常出问题，这就是抽象和具体。

作为性能测试工程师，我们要做的就是把抽象的性能需求一步步映射到具体的性能测试场景，然后运行性能测试场景，得出测试结果。

LoadRunner 的 Controller 则可帮助我们完成这个过程。

6.1　得到细化后的性能需求

从软件过程来看，软件性能需求应该是在软件项目启动之前，需求分析人员和客户共同讨论并确定下来的，这个需求对于我们性能测试人员来说，应该是明确的，无二义性的，是一系列可操作的性能指标（第 1 章中有介绍）。

但结果往往非我们所愿，如上面举例的需求二，它就不是一个明确的性能需求，需求人员应该和客户一起去细化它，最终得出真正意义上的可操作的性能指标。下面是基于业务概率计算模型的一种细化方法：

两个假定：

- 全年的业务量集中在 8 个月完成，每个月 20 个工作日，每个工作日 8 个小时；
- 采用 80-20 原理，每个工作日中 80%的业务在 20%的时间内完成，即每天 80%的业务在 1.6 小时内完成。

估算：

如果全年处理业务约 100 万笔，其中 15%的业务处理每笔业务需对应用服务器提交 7 次请求；70%的业务处理每笔业务需对应用服务器提交 5 次请求；其余 15%的业务每笔业务向应用服务器提交 3 次请求。根据以往统计结果，每

年的业务增量为 15%，考虑到今后三年业务发展的需要，测试需按现有业务量的 2 倍进行。

每年总的请求数量为：（100×15%×7+100×70%×5+100×15%×3）×2=300 万次/年。

每天的请求数量为：300/160=1.875 万次/天。

每秒的请求数量为：（18750×80%）/（8×20%×3600）=2.60 次/秒。

在正常情况下，应用服务器处理请求的能力应达到：3 次/秒。

需要注意的是，**采取何种算法、何种方法细化性能需求并不是最重要的，最重要的是软件方和客户能够达成理解一致。**

再看看需求一：

“每个用户在登录时间小于 5 秒（包含登录中下载所有资源的时间）情况下的最大用户并发数不能低于 500 个。”

这比需求二已经清楚了很多，但如果我们依照它去设计性能测试场景，会发现它同样有不够明确的问题。

是 500 个用户同时登录，响应时间小于 5 秒？还是 500 个用户分批次登录，响应时间小于 5 秒？

这在 LoadRunner 里是两个不同的场景，前者是虚拟用户数 500，同步运行；后者是增量加载方式。

好，这时客户又说话了，“我们的软件系统登录行为集中在早上八点半到八点五十之间”。这样看来，增量加载场景更符合软件系统真实的使用情况，于是我们这样设计：50 个用户一组，每分钟加载一组，10 分钟加载完 500 个用户，然后再运行场景 10 分钟，获得的响应时间要小于 5 秒。但这样还会有不能确定的疑问：

（1）第 1 分钟成功加载 50 个用户，在第 2 分钟里，第一批的 50 个用户是继

续不断执行登录的脚本还是仅仅只是保持在线状态？

（2）第一个 10 分钟结束后，成功加载了 500 个用户，然后场景继续运行 10 分钟，在这第二个 10 分钟中这 500 个用户是保持在线状态还是不断地执行其他操作？

“天哪，这种问题你问我，我怎么能知道？”估计客户也会不耐烦了。但对于我们性能测试工程师，在设计性能测试场景时，这些细节都要考虑并设置在测试场景中，因为不同的测试场景运行的测试结果会有差别，甚至很大，可能会出现运行一个场景会通过，另外一个场景却失败的情况。

可见，一份详细的软件性能需求说明是一个成功的性能测试的基础；否则可能我们辛辛苦苦得出的性能测试报告却被客户认为说明不了系统的性能情况。为了避免这样的结局，性能测试工程师应该参照本书第 1 章中的 GAME(A)模型来开展工作，在确定需求标准的时候，有以下技巧可供参考：

（1）积极参与性能需求的制定和细化工作

性能测试工程师应该比需求人员和客户更明白软件系统性能为何物，怎样度量，而且性能测试工程师还是性能测试的执行者。因此，性能测试工程师有能力、有责任去引导客户细化软件性能需求，这样性能需求来自客户，执行完性能测试，又直接反馈给客户，交流的顺畅使得性能测试的效率和质量得到提升。

（2）抓大放小设计测试场景

大多常见的性能测试属于负载测试，是模拟真实用户的集体行为。实际上，在真实环境下，用户的集体行为并不是十分明确的，甚至是混沌的，带有一定的随机性。因此负载测试与基准测试相比，在设计上并不要求十分严格，与真实业务一丝不差。基于这样的判断，设计性能测试场景可以采用抓大放小的策略，抓住用户业务行为模型的主要特征，设置场景。

在下面的章节中，我们将介绍如何依照性能需求设计性能测试场景。

6.2 设计性能测试场景

Controller 有两个视图：设计（Design）视图和运行（Run）视图。启动 Controller，在 Controller 主界面里，我们可以看到这两个视图。

1. 设计视图

设计视图显示场景中的所有 Vuser 组/脚本的列表、负载生成器（Load Generator）计算机以及分配给每个组/脚本的 Vuser 数。该视图还显示有关场景计划（手动场景）或目标（面向目标的场景）的基本信息。单击“Design”选项卡，如图 6-1 所示。

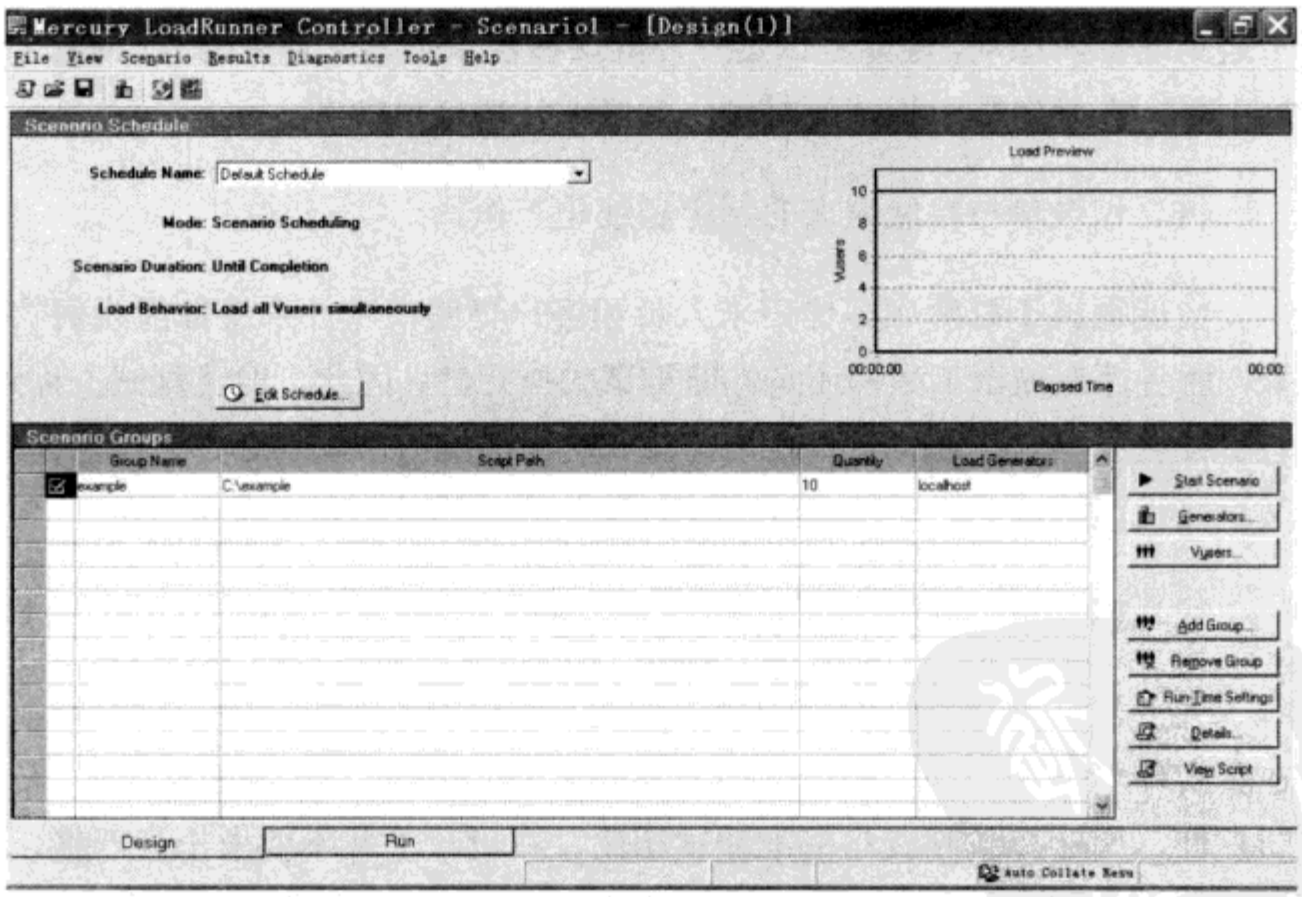


图 6-1 Controller 的设计视图

2. 运行视图

场景一旦开始运行，Controller 自动切换到运行视图，运行视图显示有关运行

的 Vuser 和 Vuser 组的信息以及联机监视器图。

单击“Run”选项卡，我们可以看到运行视图，如图 6-2 所示。

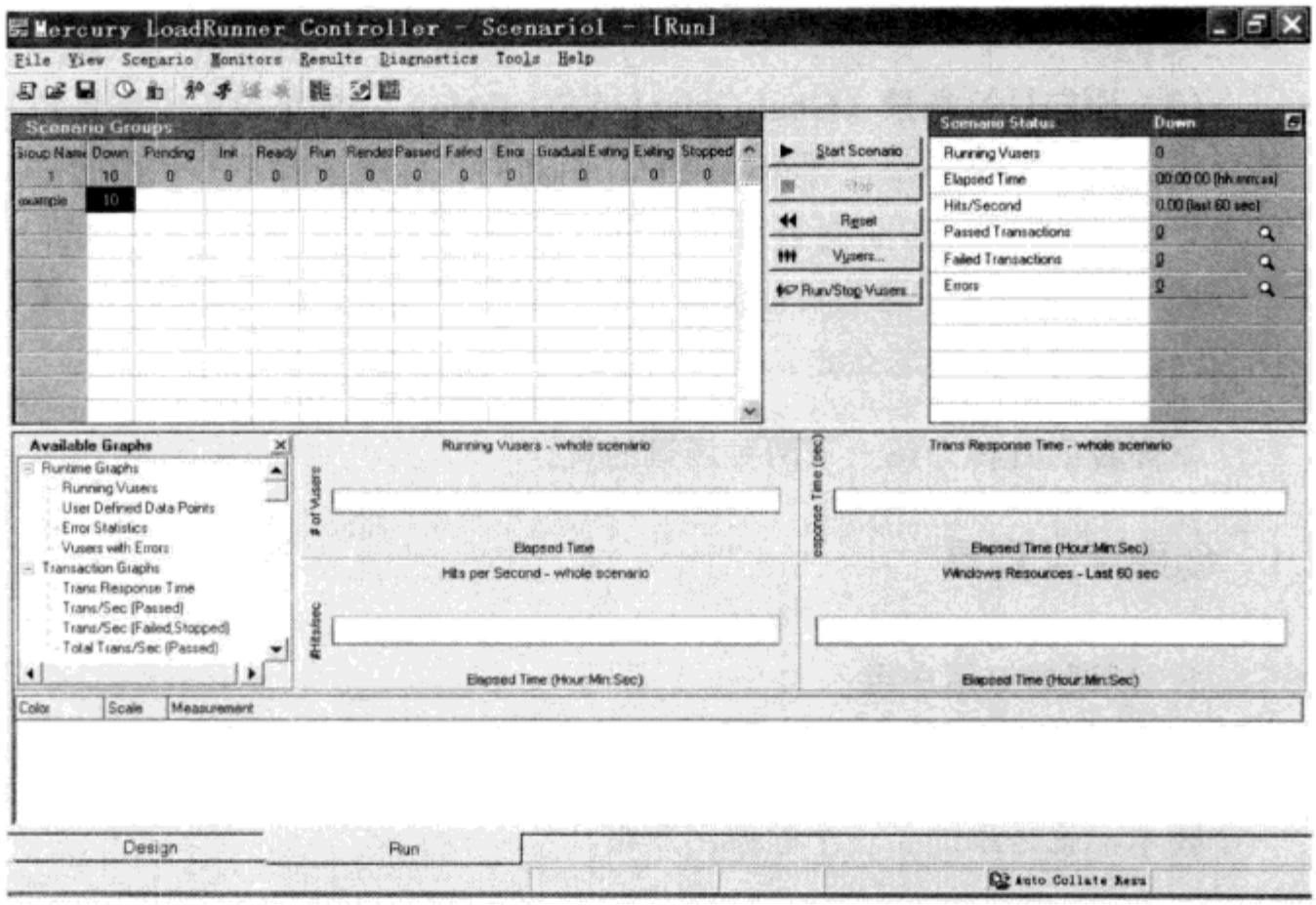


图 6-2　Controller 的运行视图

下面我们详细介绍一下怎样利用两种视图创建并监控场景。

设计性能测试场景主要是在设计视图中完成的，我们知道在性能测试中，场景的设计是十分重要的，它决定整个性能测试的成败。

在设计视图中，场景有两种类型：手工场景和面向目标场景，其中手工场景还有百分比模式。

（1）手工场景（Manual Scenario）：创建虚拟用户组，设置虚拟用户数目以及其他 Run-time 信息。

手工场景是我们常用的设计模式，负载测试就常用它。手工场景符合我们的性能测试常规思路：第一步，设置虚拟用户的数目、脚本以及它们运行的方式；

第二步，运行，得出服务器的响应时间等指标。

手工场景还有一个百分比模式：在百分比模式里，只需设定总用户数，Controller 将总用户数以百分比的方式分派去执行不同的脚本。

（2）面向目标场景（Goal-Oriented Scenario）：如果说手工场景是一板一眼的因果关系（先有条件，再得结果），而面向目标场景就是一个稍微复杂一些的闭环回馈关系了。在面向目标场景中，我们先定义测试要达到的目标，然后 LoadRunner 自动基于这些目标创建场景，运行过程中，会不断地把结果和目标相比较，以决定下一步怎么走。

下面我们具体介绍一下两种场景的设计

6.2.1 创建手工场景

1. 创建虚拟用户组

虚拟用户组是执行同一脚本的虚拟用户的集合。因此在 Controller 中，添加了一个脚本，就是添加了一个虚拟用户组。

在图 6-3 中，把脚本添加到场景操作很简单。

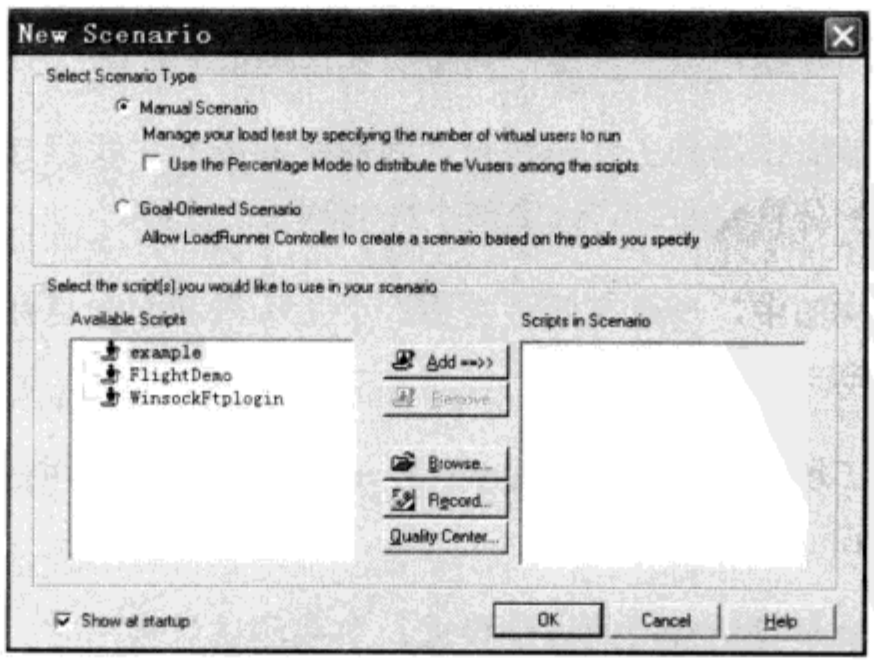


图 6-3 在 Controller 启动时，把脚本添加到场景中

如果在已经打开的场景中添加脚本，只需点击“Script Path”下的下拉箭头即可，如图 6-4 所示。

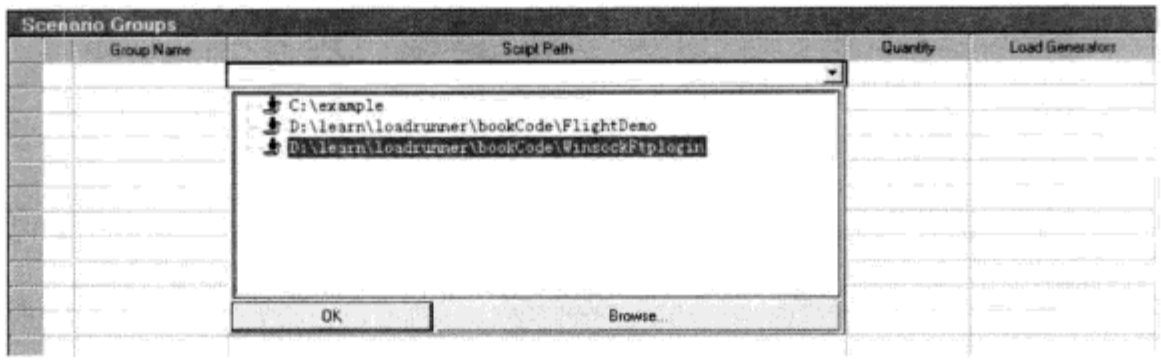


图 6-4　在 Controller 启动后，把脚本添加到场景中

2．配置虚拟用户

首先在“Quanity”下设置虚拟用户数，如图 6-5 所示。

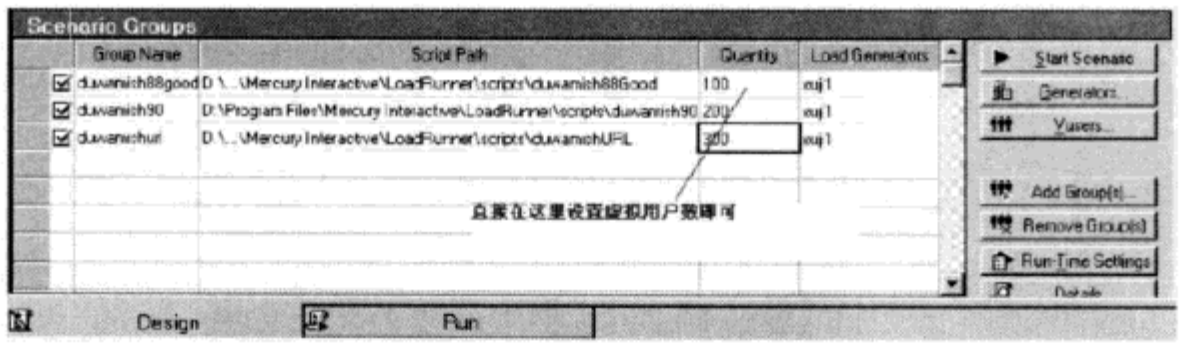


图 6-5　在场景中设定虚拟用户数量

3．详细配置虚拟用户

单击右边的“Vusers”按钮，即可详细配置每个虚拟用户的设置，包括执行的脚本、分配的 Load Generator，如图 6-6 所示。

4．设置集合点

如果在 VU 脚本中设置了集合点，Controller 默认的集合点策略是：在所有 Running 状态的 Vuser 达到集合点后才释放。如果要改变集合点策略，可按如下执行：

① 在“Scenario”菜单中选中“Rendezvous”，如图 6-7 所示。

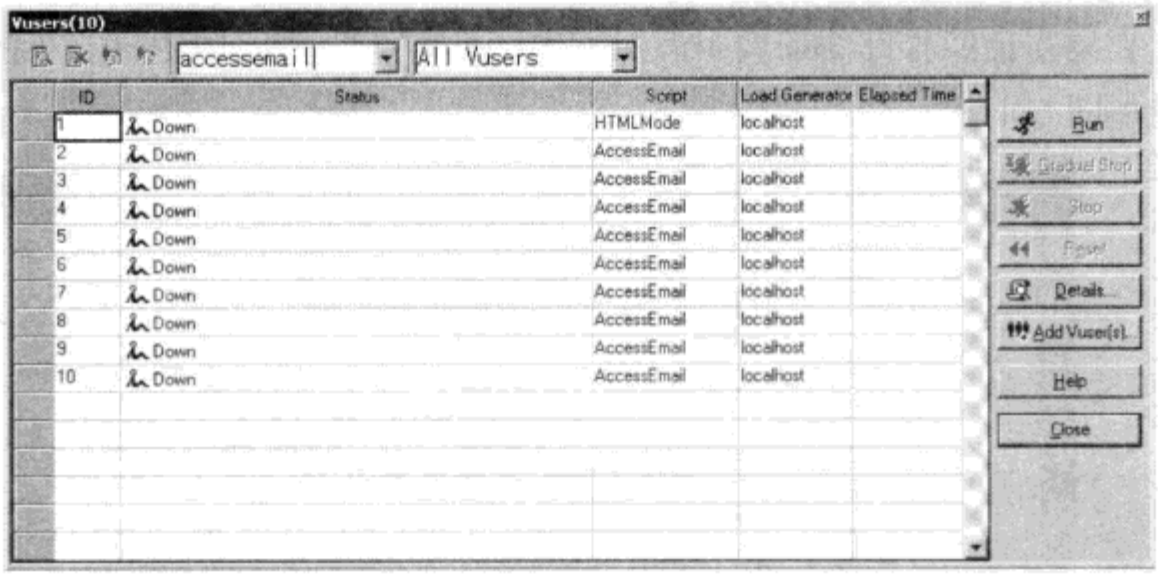


图 6-6 详细配置虚拟用户

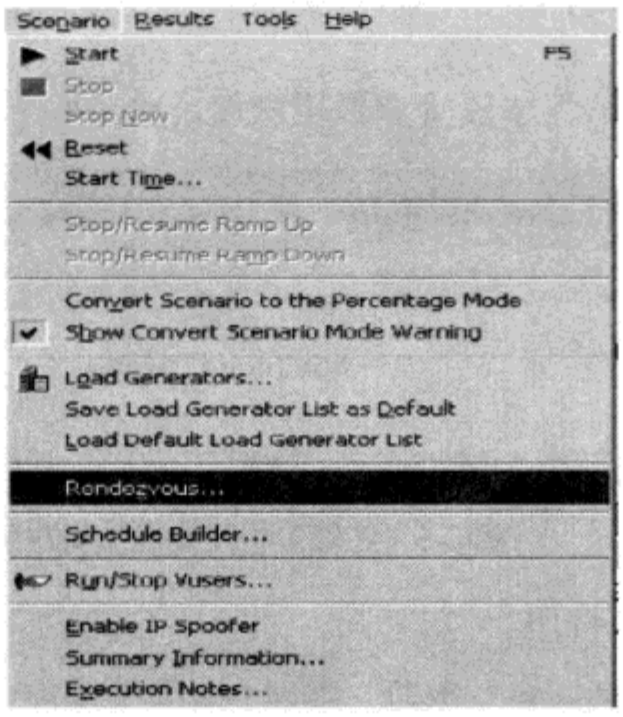


图 6-7 在“Scenario”菜单中选择设定同步点

② 打开设定同步点的详细设置对话框，如图 6-8 所示。

③ 单击 Policy... 按钮，进入策略设置窗口，如图 6-9 所示。

Policy 有三个选项：

- 第一个选项表示所有的用户到达集合点之后，再释放等待的用户，继续场景执行。

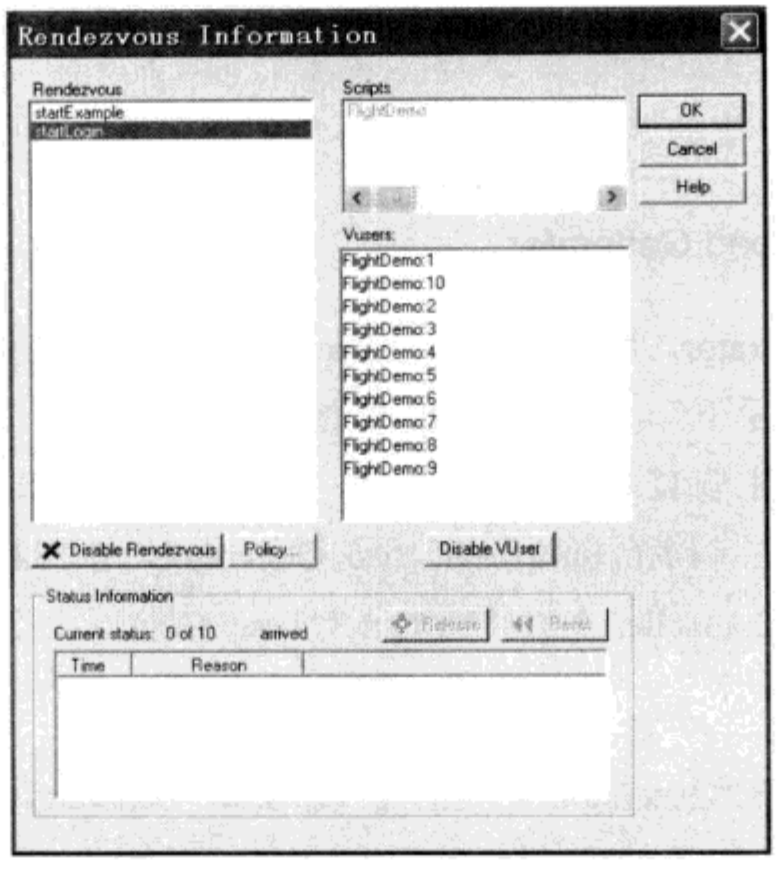


图 6-8　设定同步点的详细设置对话框

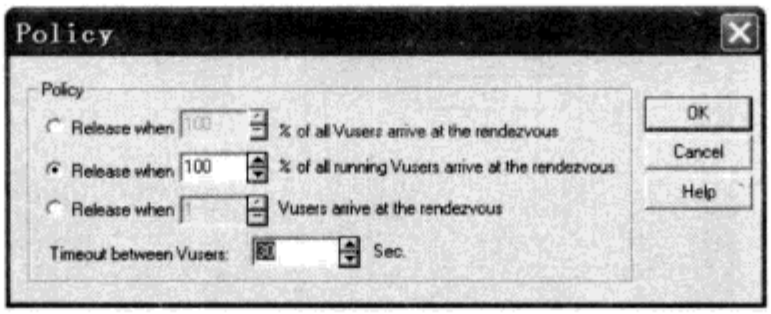


图 6-9　设定同步点策略

- 第二个选项表示所有正在运行的用户到达集合点之后，再释放等待的用户，继续场景执行。
- 第三个选项表示当指定数目的用户到达集合点之后，再释放等待的用户，继续场景执行。

Timeout 的设定表示，当第一个用户到达集合点后，等待 30 秒，如果 30 秒内上面三个选项设定的释放条件满足，就继续执行场景；30 秒后，就不再等待，开始释放等待的用户，继续执行场景。

5. 配置虚拟用户运行时设置

可参见 VU 中的“Run-time Settings”部分。

6. 配置 Load Generator

Load Generator，顾名思义，是 LoadRunner 的负载生成器，通俗点讲，就是 Controller 的“手下”，Controller 发号施令，Load Generator 负责实施和执行。通常在一台机器上安装 LoadRunner 后，就自动安装了 Load Generator，而一个 Controller 可以控制多台机器上的 Load Generator，让它们统一听从指挥，共同完成任务。那么 Controller 在给其他机器上 Load Generator 分配任务之前，首先要在网络上找到它。

单击右边的“Generators”按钮，出现“Load Generators”窗口，如图 6-10 所示。

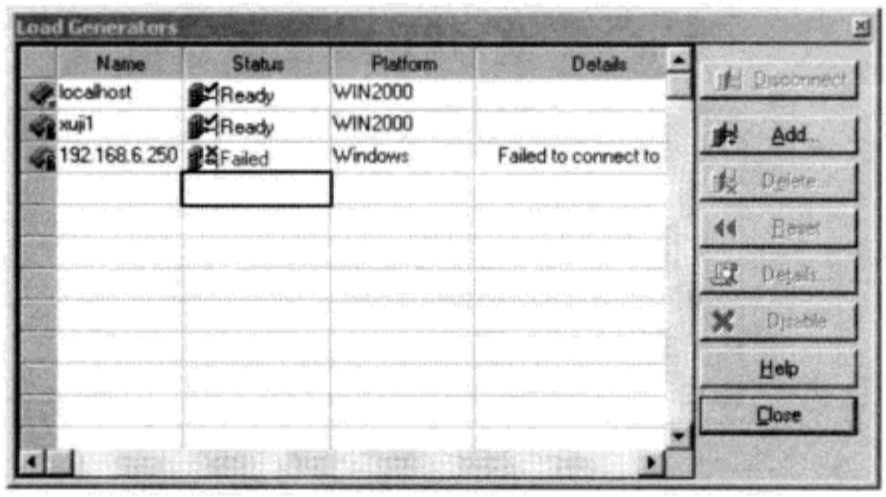


图 6-10 在 Controller 中配置 Load Generator

添加 Load Generator 后，执行“Connect”操作，使 Status 为 Ready，表示该机器连接正常；如果为 Failed，表示该机器不能连接，请检查原因。

可以把这个列表保存下来，执行菜单命令即可，如图 6-11 所示。

? 思考

在一批计算机中，我们一般选择性能最高的那台计算机做 Controller，而其他

性能稍差的可做 Load Generator，这是为什么？

7．设置网络带宽配置

默认的网络带宽配置是选择最大的网络带宽。如果在性能需求中规定了用户会通过不同的网络来使用软件系统，比如 Modem、ADSL 等，那么就需要在 Controller 的 Run-time Settings 中设置虚拟用户的带宽了。

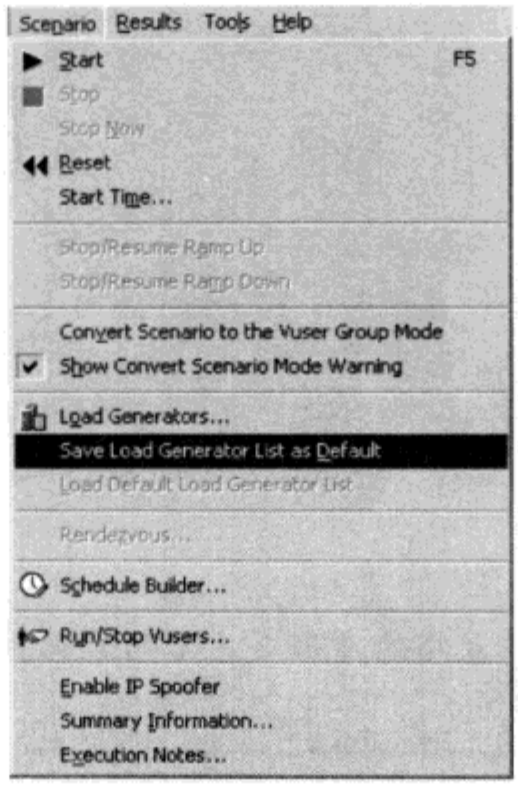


图 6-11　在“Scenario”菜单中选择保存“Load Generator”列表

使用“运行时设置”树中的“Network”(网络)下的“Speed Simulation”(速度模拟)节点，可设置测试环境的调制解调器仿真，如图 6-12 所示。

- 使用最大带宽（Use maximun bandwidth）：在默认情况下会禁用带宽仿真，并且 Vuser 按网络上的最大可用带宽运行。
- 使用带宽（Use bandwidth）：指明 Vuser 要模拟的特定带宽级别。可以选择从 14.4~512Kbps 范围内的速度，以便模拟调制解调器、ISDN 或 DSL。
- 使用自定义带宽（Use custom bandwidth）：指明 Vuser 进行模拟的带宽限制。以 bps 为单位指定带宽，其中 1Kbps=1024bps。

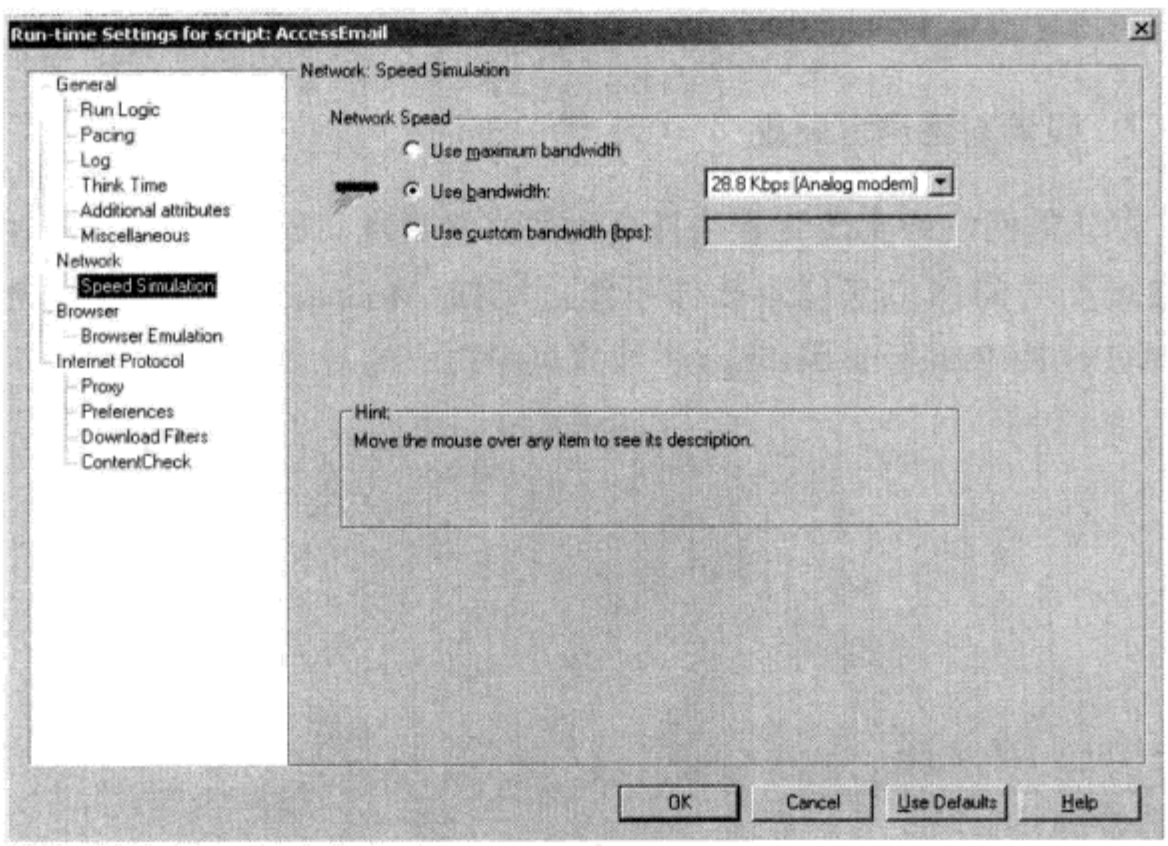


图 6-12　在 Run-time Settings 中设定带宽模拟



这里的带宽设置指的是每个虚拟用户占用的带宽，而不是多个虚拟用户共同占用的带宽。

8. 设置方案（Schedule）

这里的设置是非常重要的，主要是有关 LoadRunner 的加载策略，也是三种场景类型最重要的区别之处。单击“Edit Schedule”按钮，即可进入 Schedule 设置窗口，如图 6-13 所示。

在图 6-13 的“Ramp Up”选项卡中：

- 第一个选项是场景开始运行时，同时加载所有用户；
- 第二个选项是设定渐进加载，即按一定时间间隔增量加载虚拟用户。

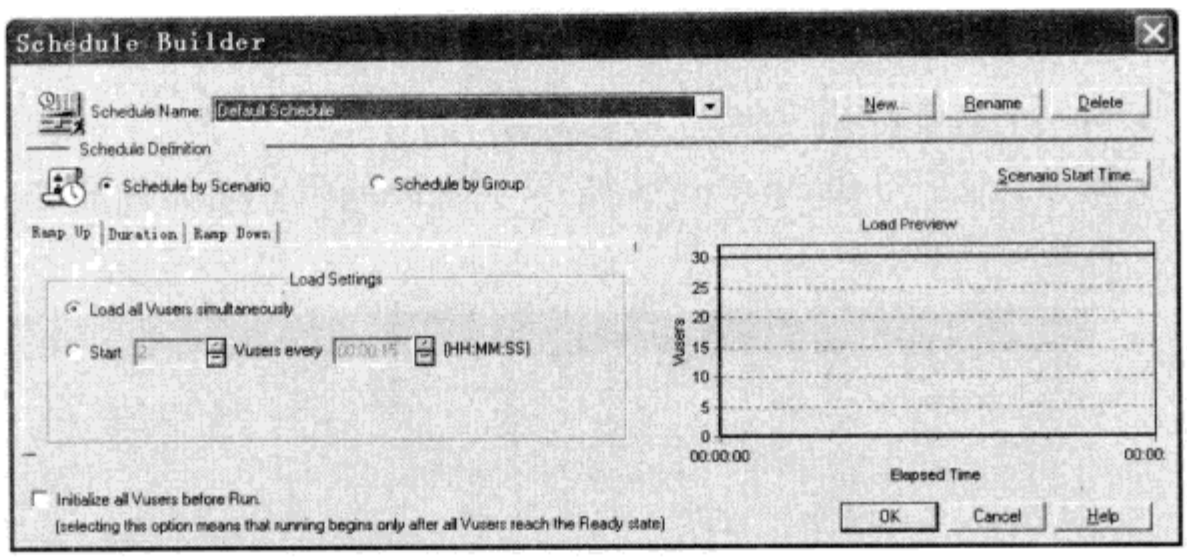


图 6-13　设定场景启动

在图 6-14 的“Duration”选项卡中：

- 第一个选项表示所有虚拟用户运行一遍脚本，就停止场景的执行；
- 第二个选项表示所有虚拟用户一直重复运行脚本，直到指定时间达到；
- 第三个选项表示一直运行，不会停止。

默认是第一个选项被勾选，这就是我们最常见的负载测试场景。

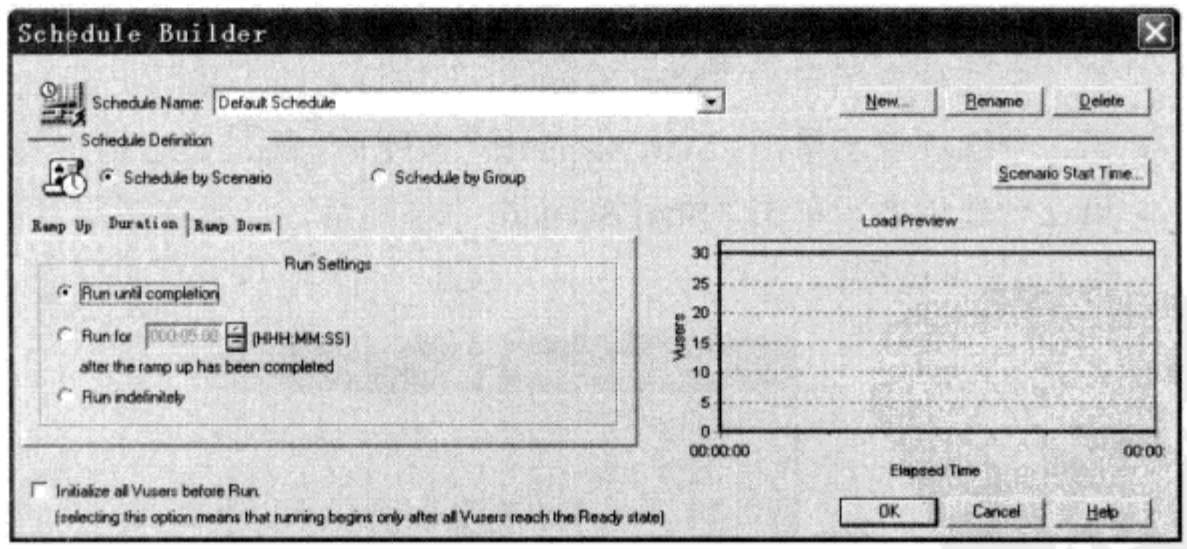


图 6-14　设定场景执行

第二个选项和第三个选项常用来做稳定性测试，比如 7×24 小时测试。

在图 6-15 的“Ramp Down”选项卡中：

- 第一个选项表示同时停止所有的用户；
- 第二个选项与 Ramp Up 类似，只不过 Ramp Down 是逐量递减虚拟用户，直到场景停止。

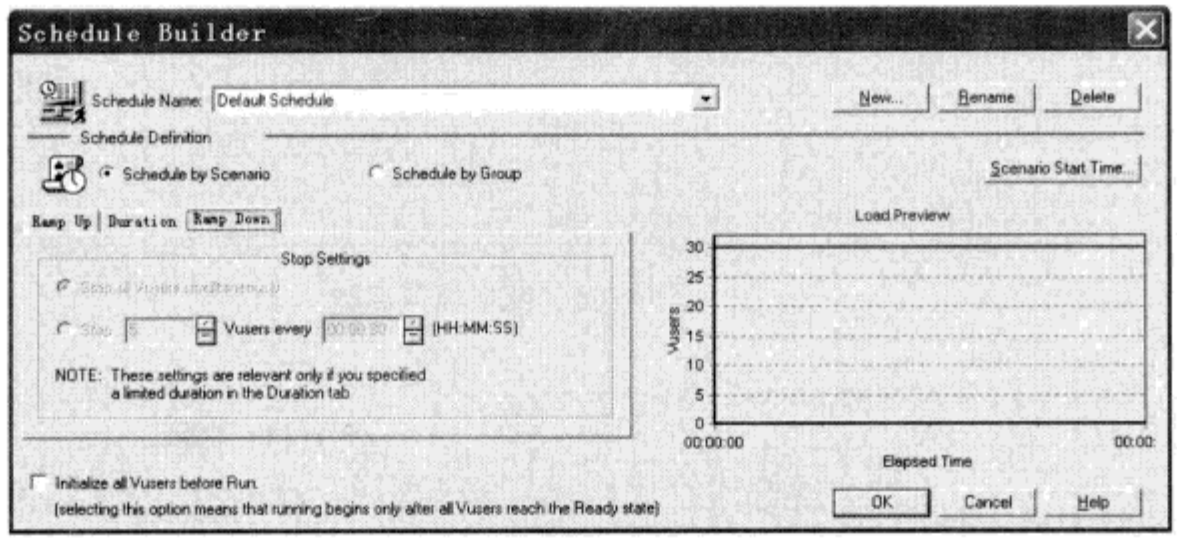


图 6-15 设定场景结束

单击 Scenario Start Time 按钮，进入 Scenario Start 窗口，如图 6-16 所示。

在图 6-16 的对话框中：

- 第一个选项表示单击“Start Scenario”按钮后，场景马上开始运行；
- 第二个选项表示单击“Start Scenario”按钮后，场景延迟指定的时间后运行；
- 第三个选项表示单击“Start Scenario”按钮后，场景在指定时刻运行，比如晚上 12 点运行。

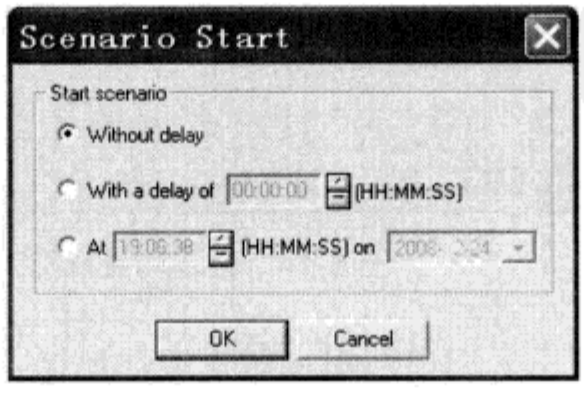


图 6-16 设定场景计划执行

6.2.2　百分比模式创建手工场景

百分比模式场景是设定总虚拟用户数，然后以百分比的形式把虚拟用户分配到各个脚本中。这种场景设置非常适合业务模型明确的性能测试。

该场景类型和“Manual Scenario”类型非常类似，下面简单地对它们不一样的地方进行设置。

在图 6-17 中，在“Total Number of Vusers”文本框里设置总虚拟用数目。

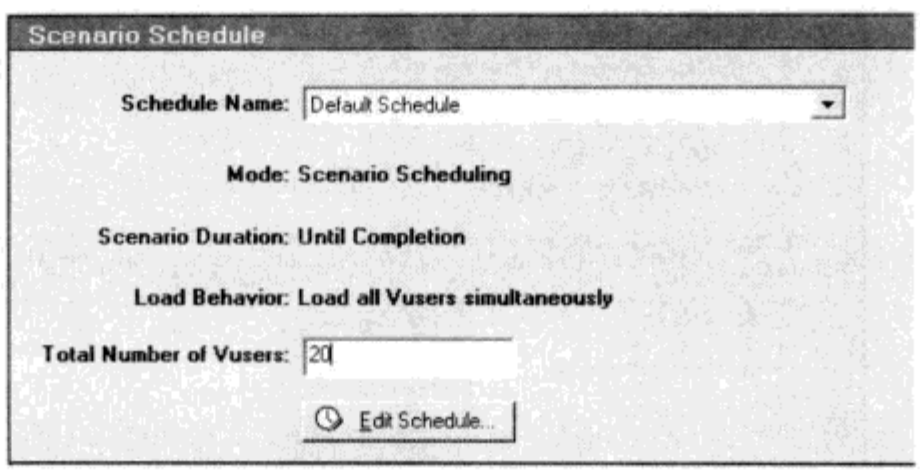


图 6-17　百分比模式的场景计划信息

在图 6-18 的“%”列里，设置各个虚拟用户组所占总虚拟用户数的百分比。

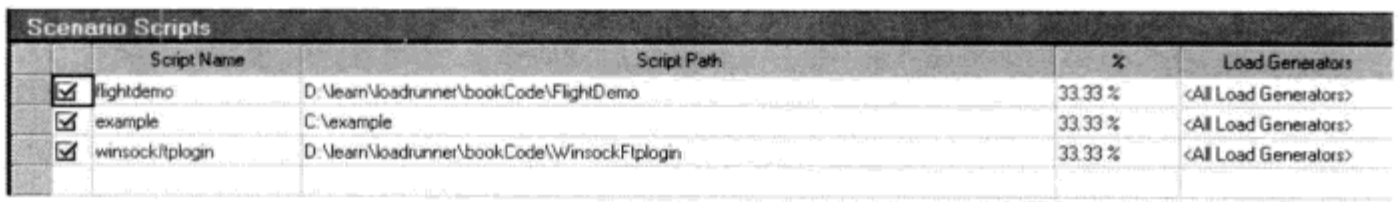


图 6-18　百分比模式的虚拟用户数目设定

? 思考

在本章开头的需求一，如果转化为手工场景，应该怎样设置？考虑两种并发情形：

（1）瞬时并发，500 个用户同时并发；

（2）增量加载，500 个用户在 2 分钟内登录系统。

6.2.3 创建面向目标场景

在前面我们已经说到面向目标场景的特点是闭环回馈关系。在面向目标场景中，我们先定义测试达到的目标，然后 LoadRunner 自动基于这些目标创建场景。

在新建场景中，选择面向目标场景（Goal-Oriented Scenario），如图 6-19 所示。

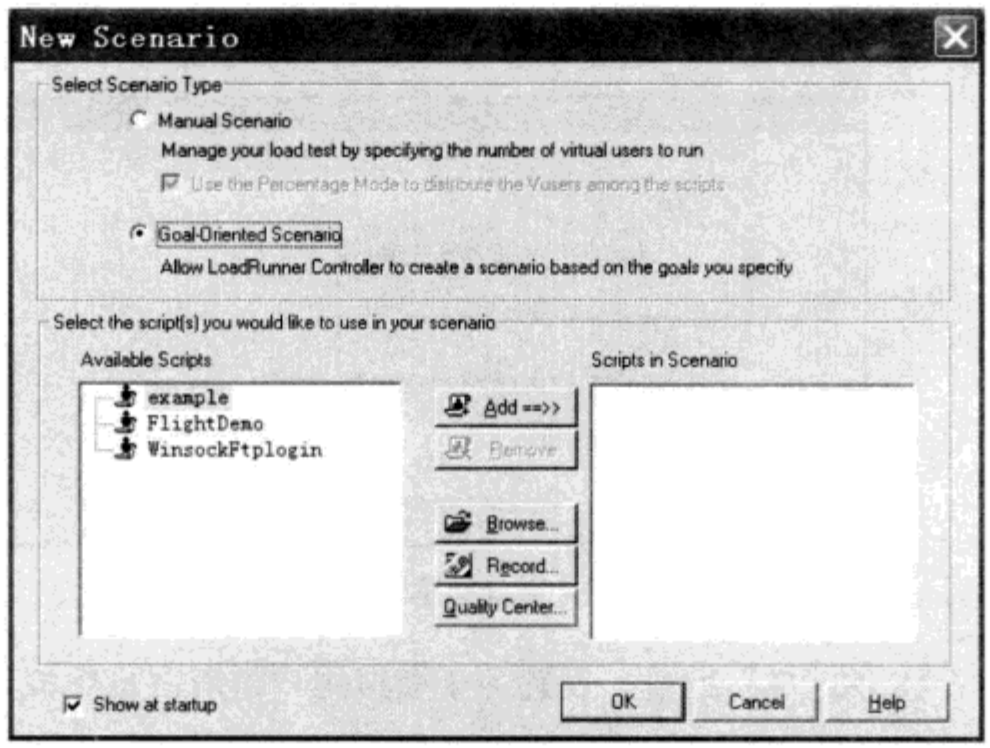


图 6-19 在 Controller 启动时选择面向目标场景

Controller 启动后，我们会看到如图 6-20 所示的页面。

单击 Edit Scenario Goal... 按钮，编辑该场景的目标，如图 6-21 所示。

在图 6-21 的“Scenario Settings”选项卡中：

- “Run Time”里设定的是一个时间，表示到达目标后，该场景会运行 30 分钟。
- “If target cannot be reached”表示的是如果目标不能达到，Controller 将会怎样处理场景，可以选择停止场景（Stop Scenario and save results），也可以选择继续运行场景（Continue scenario without reaching goal）。

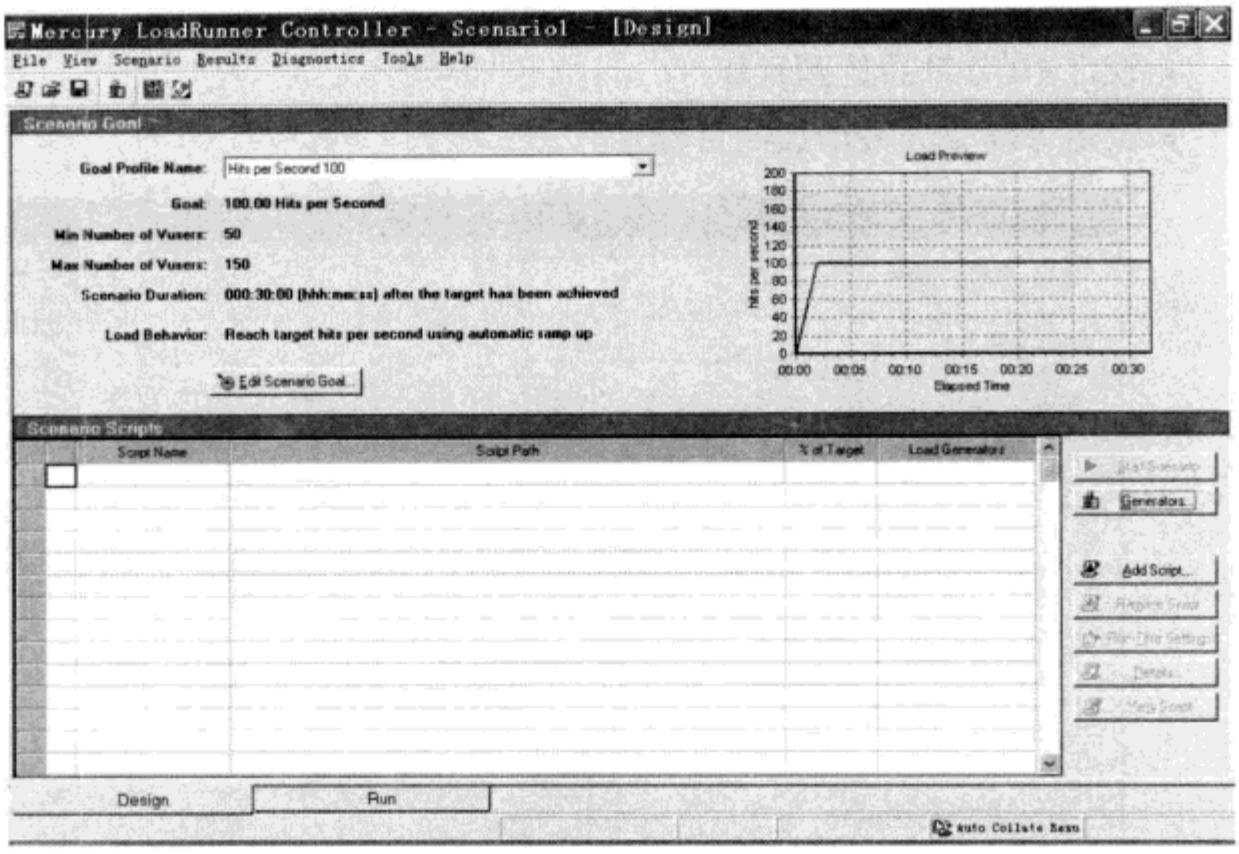


图 6-20　面向目标的场景启动后的页面

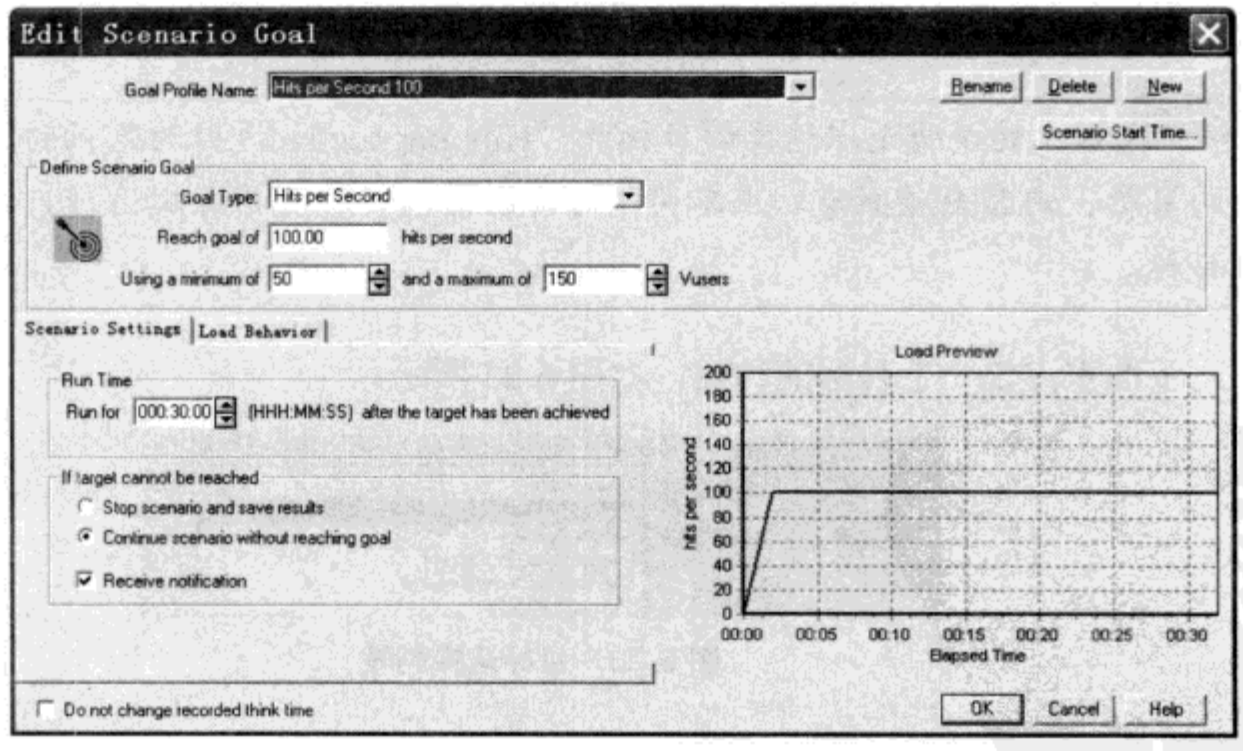


图 6-21　编辑面向目标场景的目标

在图 6-22 的“Load Behavior”设置中，“Ramp UP”有 3 个选项，分别表示：

- 让 Controller 自动加载用户；
- 设定 10 分钟后达到目标；
- 设定每隔 2 分钟增加的目标量。

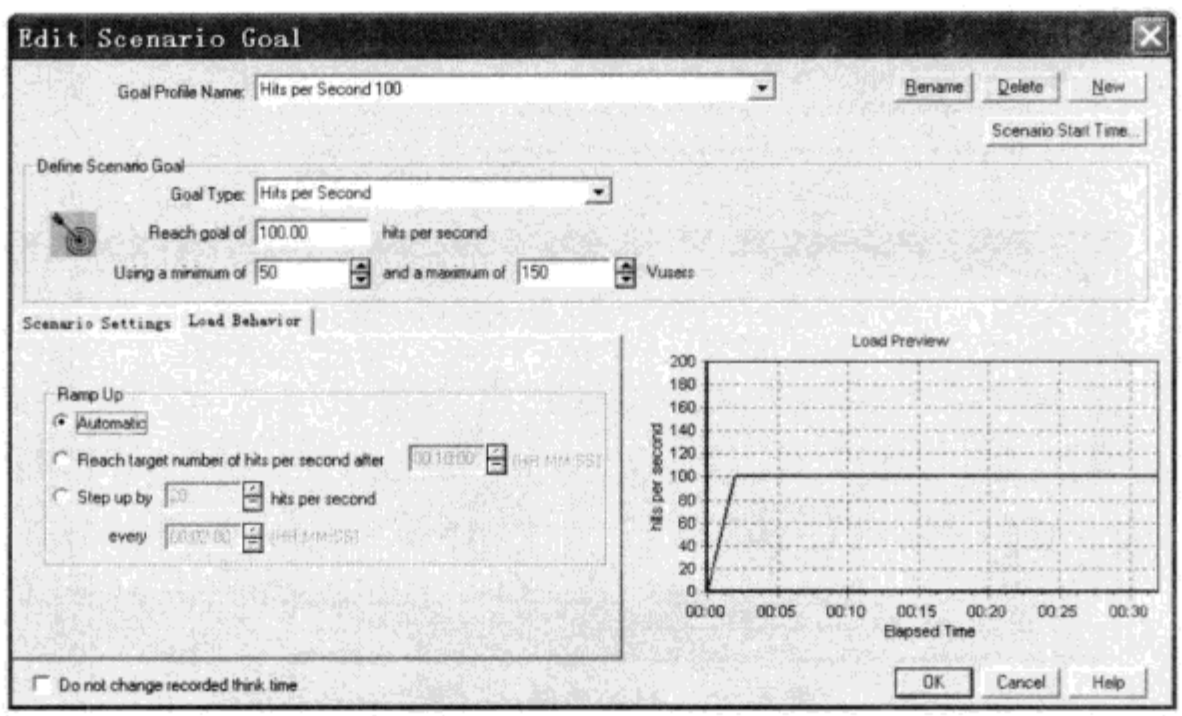


图 6-22　在面向目标场景中设定场景加载模式

提示　以上的说明是以选择的目标为“Hits per second”时为基础的。选择不同的目标，内容会稍微有一点不同。另外，面向目标场景每次运行只能设置一个目标。

下面重点说一下目标的种类，如图 6-23 所示。

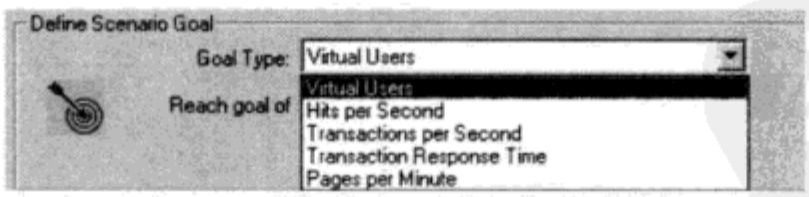


图 6-23　目标选择列表

（1）Virtual Users

如果需要测试服务器的并发处理能力，即多少人可以同时运行 Web 应用，那么推荐定义 Virtual Users 目标类型。运行定义该目标类型的场景和运行 Manual 类

型的场景类似。

具体的定义方法很简单，不再详细的说明（如图 6-24 所示）。

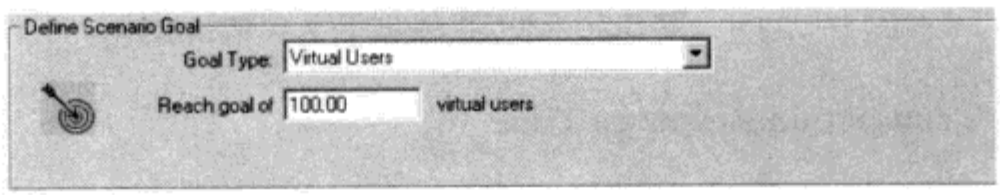


图 6-24　设定以虚拟用户数目为目标的场景

（2）Hits per Second

如果想测试 Web Server 的真正实力，推荐定义目标类型为：Hits per Second、Pages per Minute 或者 Transactions per Second，这些类型都需要指定一个虚拟用户的最小值和最大值的范围。

Controller 试图使用最少的虚拟用户来达到定义的目标。如果使用最少的用户，不能达到目标，Controller 增加用户数，直到定义的最大值。如果使用了最多的虚拟用户数，定义的目标还没有实现，那么需要增加最大用户数，重新执行场景。

在图 6-25 中，“Reach goal of”设定的是目标是 100 点击数/秒，Controller 通过调整虚拟用户数来达到目标，其中最小的虚拟用户数为 50，最大虚拟用户数为 150。

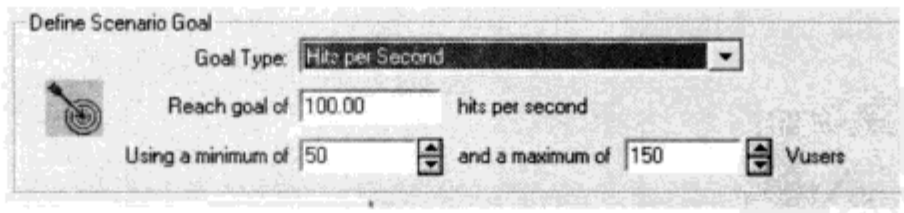


图 6-25　设定达到目标所需的最大虚拟用户数和最小虚拟用户数

（3）Transactions per Second

在图 6-26 的“Transaction Name”中应该选定目标 Transaction，当然这有一个前提，在脚本里应该有设置 Transaction，否则这里就显示为空白。

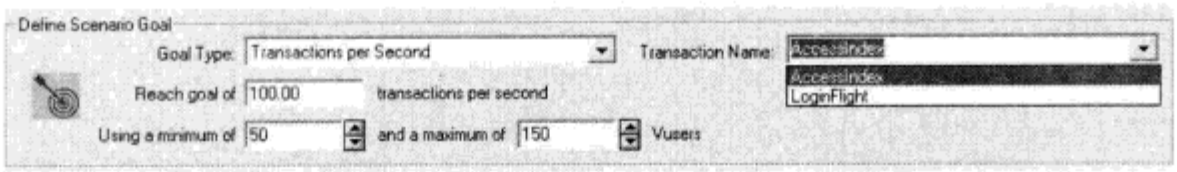


图 6-26　设定以事务为目标的场景

（4）Transaction Response Time

如果想知道在多少用户并发访问网站时，事务的响应时间达到性能指标说明书中规定响应时间的最大值，那么推荐使用“Transaction Response Time”类型。指定需要测试的事务的名称、虚拟用户数量的最小值和最大值，还有预先定义好的事务的响应时间（如图 6-27 所示）。

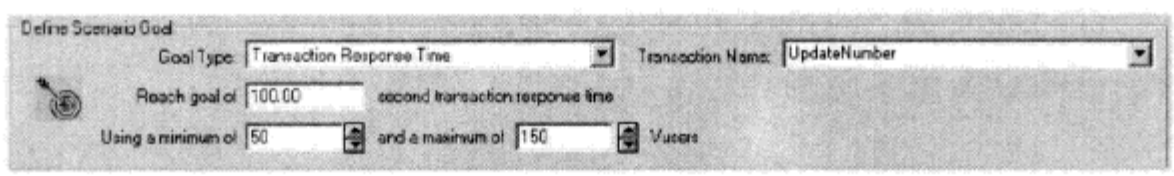


图 6-27　设定以事务响应时间为目标的场景

在场景运行中，如果使用了最多的虚拟用户，还不能达到定义的最大响应时间，说明 Web Server 还有能力接纳定义的虚拟用户的最多数量；如果使用了部分虚拟用户，就达到了定义的最大的响应时间，或者 LoadRunner 提示如果使用最多数量的虚拟用户时将要超过最大响应时间，那么需要重新设计或者修补应用程序，同时可能需要升级 Web Server 的软、硬件。

（5）Pages per Minute

各设置选项如图 6-28 所示。

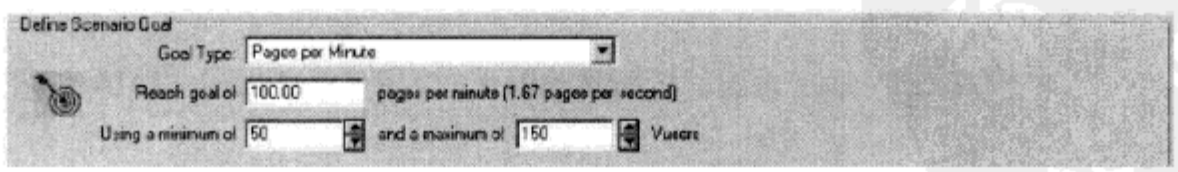


图 6-28　设定以加载页面速度为目标的场景

理解加载机制

如果你定义的类型是 Pages per Minute、Hits/Transactions per Second，Controller

首先用最小用户数除以定义的目标，得到一个值，然后确定每个用户应该达到的 Hits/Transactions 或者 Pages per Minute，然后 Controller 开始按照以下的策略加载用户：

- 如果选择的是自动加载虚拟用户，LoadRunner 会首先加载 50 个用户。如果定义的最大用户数小于 50，LoadRunner 就会一次加载所有的虚拟用户。
- 如果选择的是在场景运行一段时间后达到目标，LoadRunner 就会尝试在定义的这段时间内达到目标，根据时间限制和计算出的每个用户的 Hits、Transactions 或者 Pages，LoadRunner 确定第一批加载多少用户。
- 如果选择的是按照一定的阶段达到目标（也就是先在 x 长时间内达到 y Pages/Hits，然后再达到下一个目标），LoadRunner 计算每个用户应该达到的数字后，再确定第一批加载多少用户。

每加载一批用户后，LoadRunner 会判断是否达到这批用户的目标。如果这批用户的目标没有达到，LoadRunner 重新计算每一个用户应该达到的目标数字后，重新调整下一批加载用户的数量。在默认情况下，LoadRunner 每 2 分钟加载一批用户。

如果 Controller 加载了最多数量的用户还没有达到预定的目标，LoadRunner 会重新计算每个用户的目标，然后同时运行最大数量的用户，尝试达到预定的目标。

如果出现以下情况，Pages per Minute、Hits/Transactions per Second 类型的场景会置于“Failed”状态：

- Controller 使用了指定的最大数量的用户，并且两次都没有达到目标；
- 所有的用户运行都失败；
- 没有足够的 Load Generators 机器（现有的机器已经超载运行的情况下）；
- Controller 增加了几批用户后，Pages per Ninute 或者 Hits/Transactions per Second 没有增加。



在面向目标场景中，集合点是不可用的。

6.2.4 多 IP 的实现原理以及模拟

当运行场景时，虚拟用户使用它们所在的 Load Generator 的固定的 IP 地址。同时每个 Load Generator 上运行大量的虚拟用户，这样就造成了大量的用户使用同一 IP 同时访问一个网站的情况，这种情况和实际运行的情况不符，并且有一些网站会根据用户 IP 来分配资源，这些网站会限制同一个 IP 的登录、使用等。为了更加真实地模拟实际情况，LoadRunner 允许运行的虚拟用户使用不同的 IP 访问同一网站，这种技术称为“IP 欺骗”。

启用该选项后，场景中运行的虚拟用户将模拟从不同的 IP 地址发送请求。该选项非常有用。



IP Spoofer 要在连接 Load Generator 之前启用。

要使用 IP 欺骗，各个 Load Generator 机器必须使用固定的 IP，不能使用动态 IP（即 DHCP）。

使用 IPSpoofer 的步骤如下：

（1）使用 IP Wizard：在“开始”菜单中，找到“LoadRunner”>“Tools”>“IP Wizard”运行它，如图 6-29 所示。

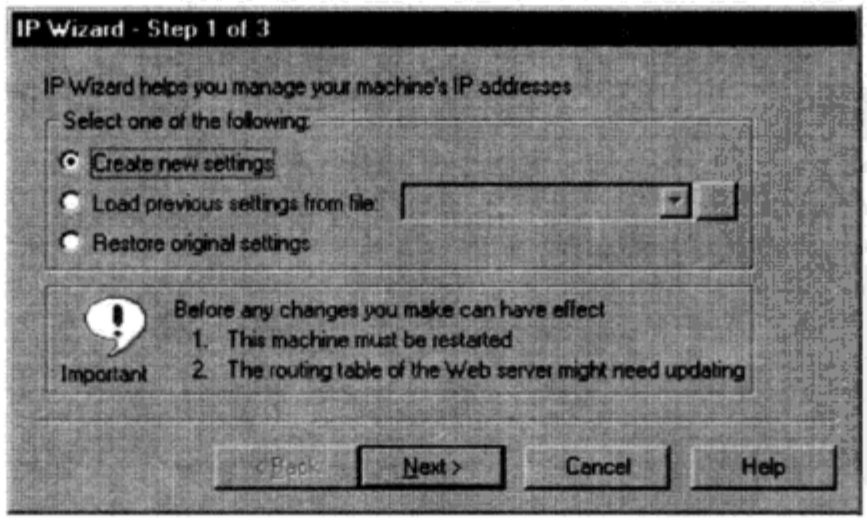


图 6-29 IP 欺骗设定首页面



注 意　运行 IP Wizard 程序的机器必须使用固定的 IP，不能使用动态 IP。

第一次运行 IP Wizard，需要选择第一项“Create new settings”；如果以前运行过，可以选择第二项“Load previous settings from file”，选择保存好的文件；第三项用于使用 IP 欺骗进行测试完成后，释放 IP 的过程（因为该机器会占用大量的 IP 资源，可能会导致其他机器没有 IP 可用的尴尬局面，使用该项，可以恢复到原来的状况）。这里选择第一项，单击“Next”按钮，出现 IPWizard 的第二个窗口，如图 6-30 所示。

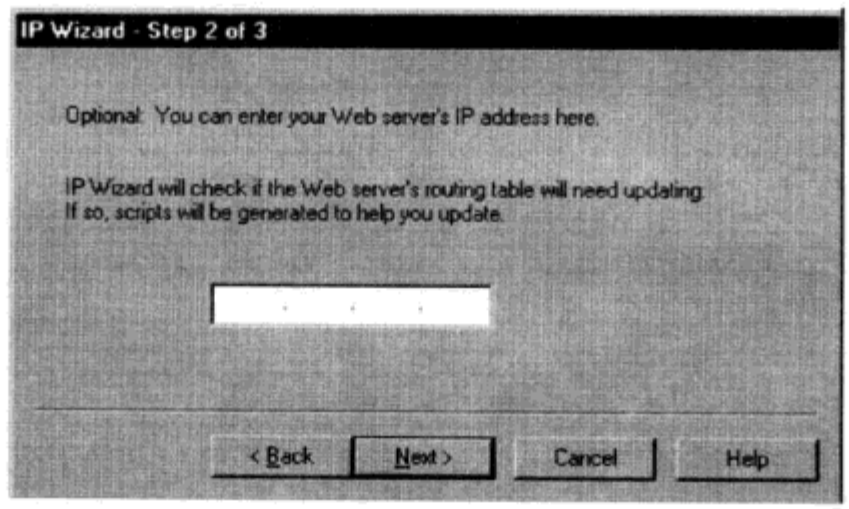


图 6-30　IP 欺骗——设定 Web Server 的 IP

这里输入 Web Server 的 IP 地址，然后单击“Next”按钮，出现向导的第三个窗口，如图 6-31 所示。

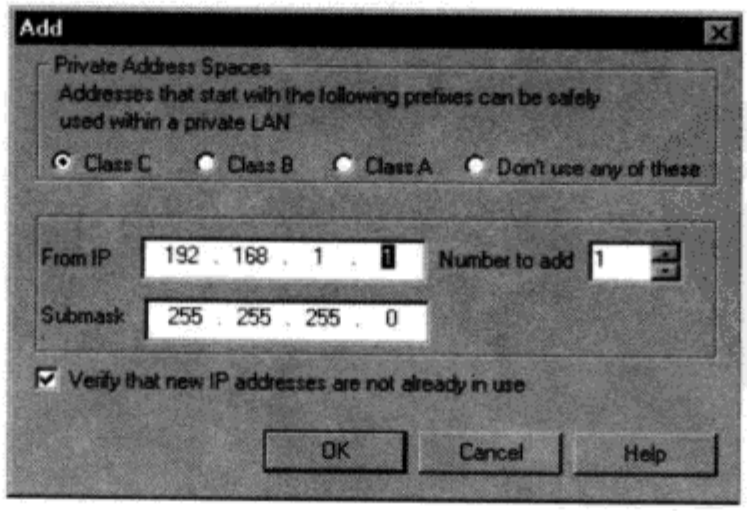


图 6-31　IP 欺骗——设定虚拟 IP 段和子网掩码

在“From IP”文本框中输入要使用 IP 范围的第一个 IP 值，然后在“Number to Add”中输入一个数字，表示 IP 范围的值；假如第一个 IP 为 192.168.6.100，范围大小为 100，那么 IP Wizard 将会使用 192.168.6.*N*（100<=*N*<200），当然这个范围内已经使用的 IP 地址除外，否则会引起 IP 冲突。“Submask”采用默认设置即可，取决于使用哪种类型的网络 IP，一般局域网内采用 Class C 即可。

单击“OK”按钮，IP Wizard 开始检查该范围内有没有使用的 IP，并把没有使用的 IP 添加到本机的 IP 窗口中。到最后一个窗口，直接单击“Finish”按钮，使用 IPWizard 后，需要重新启动机器。

(2)在 Controller 的场景中，启用 IPSpoofer 即可。启用方法很简单，在 Controller 的“Scenario”菜单中选中“Enable IP Spoofer”，如图 6-32 所示。

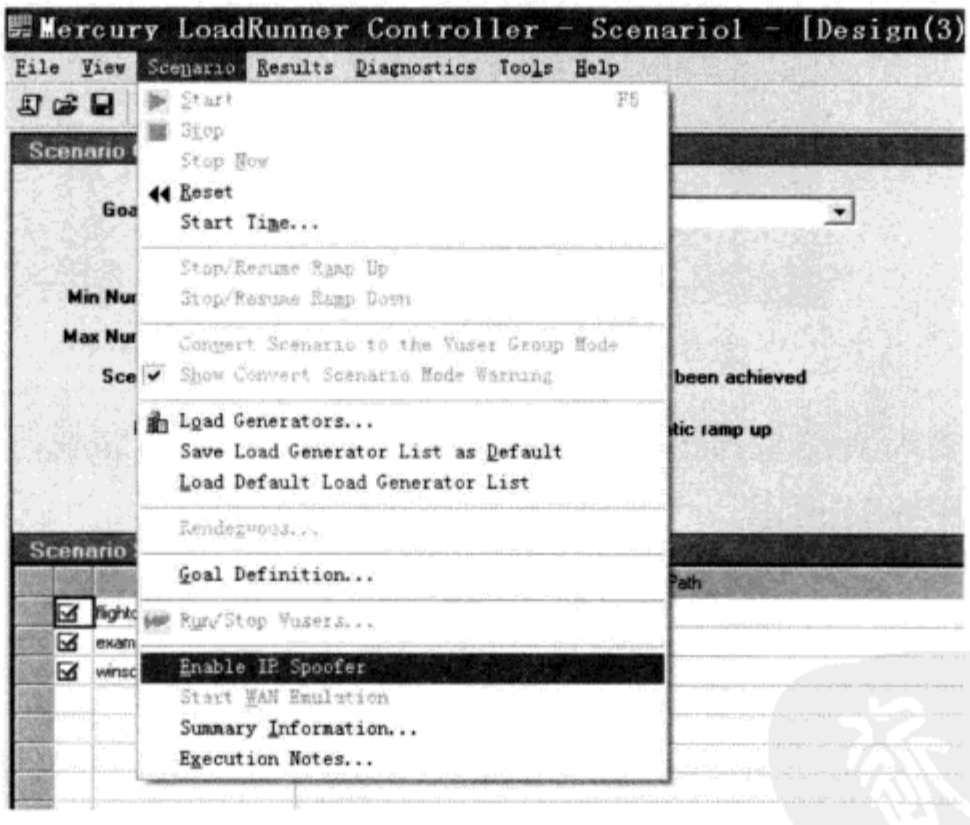


图 6-32　在 Controller 中激活虚拟 IP 设置

查看虚拟 IP 是否实现：

可以在 VU 脚本中使用 lr_get_vuser_ip 函数来得到当前虚拟用户的 IP 地址，在脚本中加入如下语句：

```
char *ip;
ip = lr_get_vuser_ip();
if (ip)
    lr_output_message("The IP address is %s", ip);
else
    lr_output_message("IP spoofing disabled");
```

场景运行时，如果虚拟 IP 生效的话，打开每个 Vuser 的日志，就可以看见他们各自的 IP 了。

6.3　运行场景

性能测试场景的运行和监控都是在运行视图中进行的。

在本节中，我们按照两条线索、三个对象来看看 Controller 是怎么运行场景的。两条线索是场景的控制和场景的查看，三个对象是场景（Scenario）、Vuser 组和 Vuser。

6.3.1　场景控制

在前面，我们主要说的是场景的设置，设置结束后，就可以运行场景了。在本小节的场景控制里，我们将要学习怎样运行场景，以及运行期间的控制。

“运行”选项卡显示“场景组”窗口、场景状态摘要和在脚本执行期间生成的联机信息的图。

场景组：显示每个虚拟用户组及其当前运行状态，如图 6-33 所示。

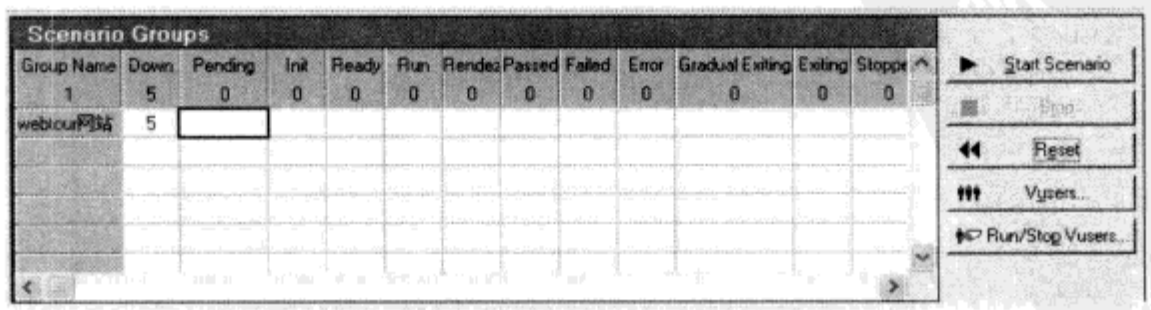


图 6-33　场景组信息

在场景组的左边区域显示虚拟用户组的状态，右边区域则是场景的控制，比如开始、停止、复位、虚拟用户控制等。

- ▶ Start Scenario（开始场景）：这是我们性能测试的“点火”按钮，单击它意味着性能测试场景开始运行。这时 Controller 开始初始化场景中的 Vuser 并且将它们分配给指定的负载生成器，即它们开始执行其 Vuser 脚本。



如果我们在场景设置中设了场景开始时间，单击“Start Scenario”按钮，只是开始计时，只有到了指定的时间，Controller 才开始运行场景。

- ■ Stop（停止场景）：在单击了“Start Scenario”按钮后，“Stop”按钮由灰变亮。我们在场景运行期间的任何时候，单击它就可以停止场景的运行。但停止的方式有三种：
 - ➢ 当前迭代结束后，再停止运行场景（Wait for the current iteration to end before stopping）；
 - ➢ 当前 Action 结束后，再停止运行场景（Wait for the current action to end before stopping）；
 - ➢ 无须等待，立即停止运行场景（Stop immediately）。

这些停止方式在场景运行之前就需要在“Tools”菜单下的“Options”选项中设置好。如图 6-34 所示，选中 Run-time Settings 选项卡。

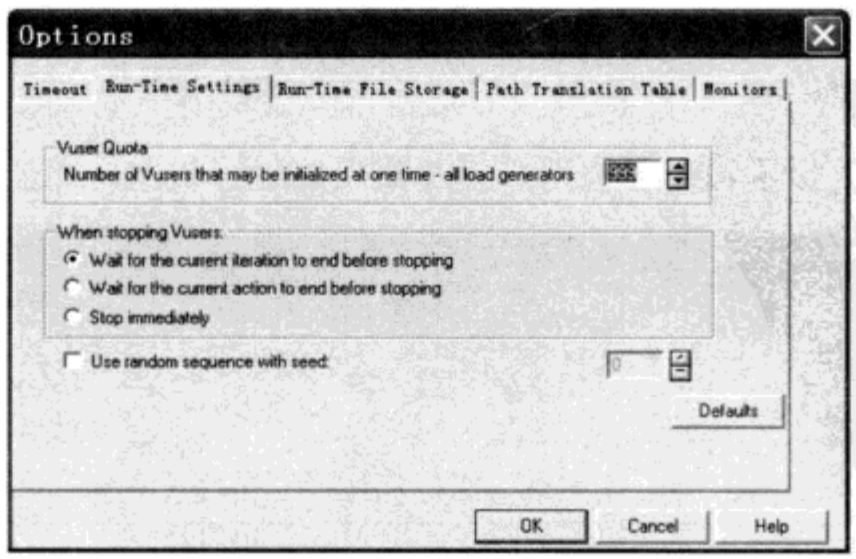


图 6-34 设置场景停止方式

如果我们之前选中的是第三个选项，即“立即停止”选项，则在单击“Stop”按钮之后，场景中的所有 Vuser 都将变为“正在退出”状态。如果我们之前选中的是“退出前等待当前迭代结束”或者“退出前等待当前 Action 结束”选项，则场景中的 Vuser 都将变为“逐步退出”状态，全部 Vuser 退出后，场景再停止。

- Stop Now（立刻停止）：指示 Controller 立即停止运行方案。作用同上面的第三个停止方式——立即停止（Stop immediately）。
- Reset（重置）：将方案中的所有 Vuser 组重置为其方案前的“关闭”（Down）状态，即回到原位，准备下一次场景执行。
- Vusers...（虚拟用户组）：打开“Vuser”对话框，可以查看 Vuser 组中每个 Vuser 的详细状态。
- Run/Stop Vusers...（运行/停止 Vusers）：打开“运行/停止 Vuser”对话框，可以在场景运行期间决定是继续执行还是停止某个用户组。

通常，我们通过以上几个按钮操作就能对场景做简单的控制了。但它们的操作是对整个场景都起作用，当场景中包含多个脚本，即多个虚拟用户组的时候，我们可能要分别控制各个虚拟用户组的行为了。

1．控制整个虚拟用户组

您可以对方案（Schedule）中的某个 Vuser 组执行下列操作：通过在“场景组”（Scenario Groups）窗口中右键单击，弹出如图 6-35 所示的下拉菜单。

- 重新编号（Renumber）：重新对组中的 Vuser 编号，从而更改它们的 ID 号。ID 号是场景运行时分配给虚拟用户组的各个 Vuser 的。一般的，Vuser 执行的结果和其 ID 没有联系。如果在某个场景下，30 号的 Vuser 总是执行出错，那么就考虑重新编号试试。
- 初始化组（Initialize Group/s）：将组中的 Vuser 分配给其指定的负载生成器，这样它们就可以执行它们的脚本了。Vuser 组的状态将从“关闭”变为“挂起”、“正在初始化”、“就绪”。如果 Vuser 组初始化失败，该

Vuser 组的状态将变为“错误”。



图 6-35 右键单击虚拟用户组展现的控制菜单

通过在运行组中的所有 Vuser 之前将这些 Vuser 都初始化，可以确保它们能够同时开始执行方案。

- 运行组（Run Group/s）：此 Vuser 组开始执行其脚本。如果运行处于“关闭”或“错误”状态的 Vuser 组，LoadRunner 将初始化该 Vuser 组，然后再运行它。在多个虚拟用户组的场景里，我们可以使用这个命令去单独运行某个虚拟用户组，从而查看单个虚拟用户组对系统性能的影响。
- 运行一个 Vuser（Run One Vuser）：指示 Controller 随机运行 Vuser 组中的某个 Vuser。注意，将打开 Vuser 日志，显示有关 Vuser 的运行时信息。
- 暂停（Pause）：临时暂停此虚拟用户组的执行。Vuser 组的状态将从“正在运行”变为“暂停”。

注意 暂停 Vuser 组将影响它的事务响应时间。

- 逐渐停止（Gradual Stop）：指示 Controller 在停止 Vuser 组之前完成当前的迭代或操作。注意，如果已经在“选项”对话框的“运行时设置”选项卡中选择了“退出前等待当前迭代结束”或者“退出前等待当前操作结束”

选项，则此选项仅在 Vuser 组处于运行状态时可用。

- 停止（Stop）：指示 Controller 立即停止运行 Vuser 组。
- 重置组（Reset Group/s）：将组中的所有 Vuser 都重置为其方案前的“关闭”状态。
- 启用（Enable）：使 Vuser 组参与场景。
- 禁用（Disable）：禁用 Vuser 组，以便它不再参与场景。
- 显示 Vuser（Show Vuser）：为组中的每个 Vuser 打开运行时查看器。
- 隐藏 Vuser（Hide Vuser）：关闭打开的运行时查看器。
- 显示 Vuser 日志（Show Vuser Log）：打开包含组中每个 Vuser 的运行时信息的脚本日志。在默认情况下，Vuser 脚本日志每 1000 毫秒刷新一次。
- 隐藏 Vuser 日志（Hide Vuser Log）：关闭 Vuser 脚本日志。
- 按名称排序（Sort By Name）：按名称的字母顺序对组进行排序。

2. 控制个别 Vuser

要控制单个 Vuser，请执行下列操作：

选择 Vuser 组，然后单击“Vuser”按钮，将打开“Vusers”对话框，其中显示该组中每个 Vuser 的 ID、状态、脚本、负载生成器和已用时间（自方案开始）的列表，如图 6-36 所示。

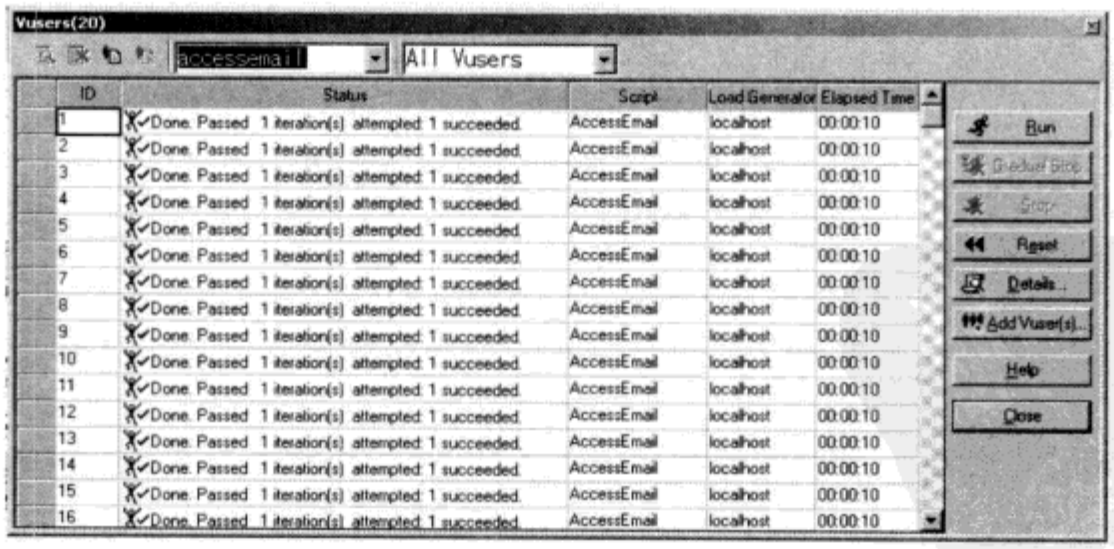


图 6-36　虚拟用户运行详细信息

我们从这个对话框里可以看清楚哪个虚拟用户慢，哪个快，它们各自处于什么状态。在特定的情景下，我们可以直接控制单个 Vuser，让它运行或退出，比如

有的虚拟用户已经好久都没响应了，为了不让它影响其他在集合点等待的虚拟用户，就不对它吹“集结号”了，直接通过“Stop”命令停止它。

可以使用下列实用程序来控制单个 Vuser：

- 选择 Vuser 并单击“运行”（Run）按钮，可以运行它。
- 选择 Vuser 并单击“停止”（Stop）按钮，，可以使它立即停止运行。

右键单击某个 Vuser，在弹出的右键下拉菜单里，还有更详细的控制功能，如图 6-37 所示。



图 6-37　右键单击单个虚拟用户弹出的控制菜单

这个菜单的一些功能和虚拟用户组的右键菜单功能很类似，可作为参照，在此不做赘述。

下面介绍在单个 Vuser 控制菜单中一些比较特殊而有用的功能：

- 要查看正在执行所分配的脚本的 Vuser，选择该 Vuser 并单击“显示”（Show Vuser），这时会打开 LoadRunner 的运行时查看器（Run-Time Viewer），如图 6-38 所示。我们在 VU 一章已经介绍到这是一个简约版本的浏览器，在运行时查看器里，我们会看到当前 VU 执行脚本时服务器返回的页面，这是非常直观的。



图 6-38　在场景运行时打开运行时查看器来查看脚本执行状态



提示　在默认情况下，运行时查看器只能解析和显示 HTML 页面，如果页面中含有 Java Script、Java Applet 或 ActiveX 控件，要在运行时查看器的选项（Options）菜单下勾选相关设置，如图 6-39 所示。

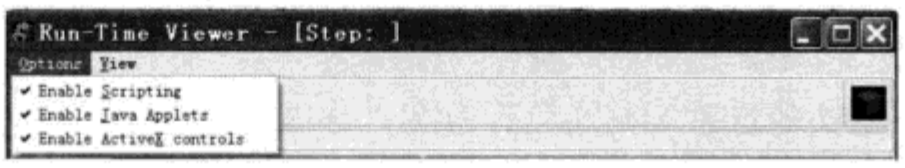


图 6-39　运行时查看器中设置展现控件选项

- 要关闭运行时查看器，请单击“隐藏”（Hide Vuser）。
- 要查看 Vuser 脚本日志，请单击“显示 Vuser 日志”（Show Vuser Log），将显示如图 6-40 所示的脚本日志。注意，要在这里显示日志，应该在 Run-time Settings 中选择相应的日志等级。

（1）高级——从同步点中手动释放 Vuser

在运行方案时，在 Controller 释放 Vuser 之前可以从集合中手动释放它们。

要从集合中手动释放 Vuser，请执行下列操作：

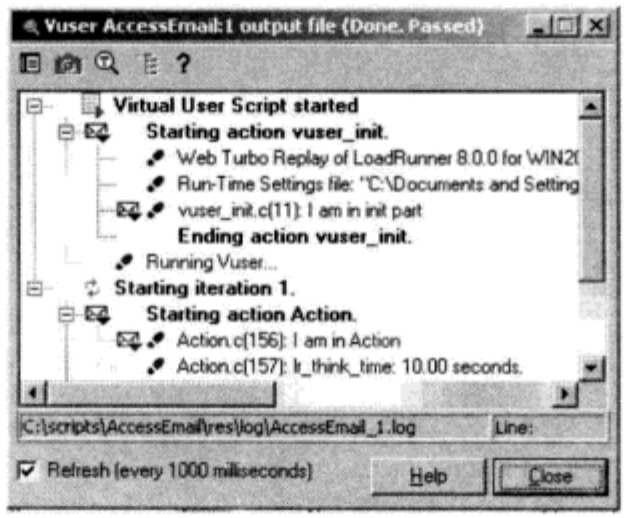


图 6-40　在 Controller 中查看虚拟用户执行脚本详细日志

① 选择“Scenario”（场景）>“Rendezvous”（集合点），将打开“同步点信息”对话框。

② 从集合列表中选择集合。

③ 单击“释放”按钮，将释放该集合中的 Vuser。

（2）高级——向正在运行的场景中手动添加 Vuser

在运行场景期间，使用“运行/停止 Vuser”对话框可以手动控制新 Vuser 的添加。该对话框因所运行的场景模式而异：

- 如果是以普通手工方式运行方案，你能控制可以添加到每个 Vuser 组中的新 Vuser 数，以及将运行这些附加 Vuser 的负载生成器。

单击场景组的“Vuser”按扭，然后在弹出的虚拟用户列表对话框中单击右侧控制区域的“Add Vuser”按钮，在如图 6-41 所示的对话框中，设置新添加的虚拟用户信息。

- 如果是以“百分比模式”运行方案，则你能控制可以根据定义的百分比在 Vuser 脚本中分配的新 Vuser 数，以及将运行这些附加 Vuser 的负载生成器。

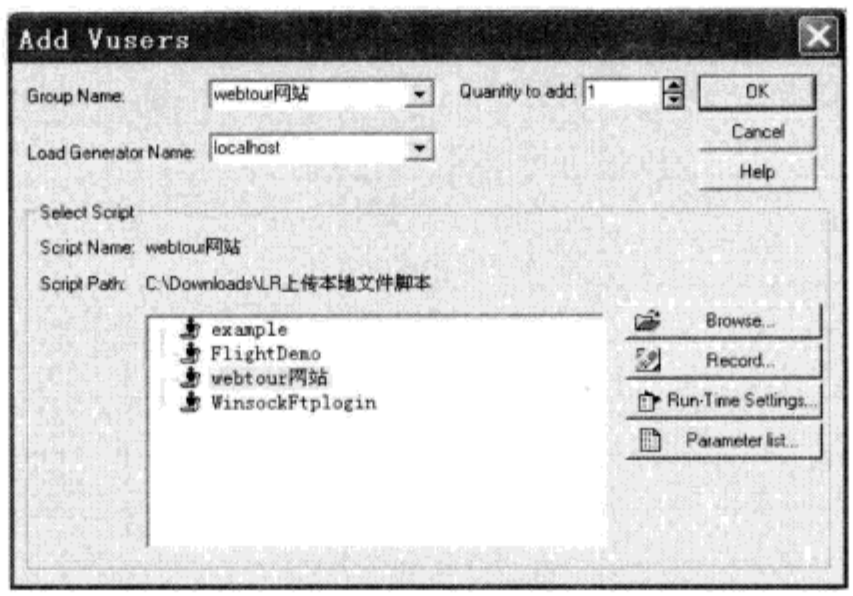


图 6-41　场景运行时手工添加虚拟用户

6.3.2　执行期间查看场景

1．查看用户组（Vuser Group）状态

在上一节，我们学习了怎样启动、停止场景等一些基本的操作，这一节我们学习怎样看懂场景里的信息，怎样了解和查看场景当前的执行状态。

场景执行期间，“场景组”的窗格中会显示各个虚拟用户组（Vuser Group）的状态，如图 6-42 所示。

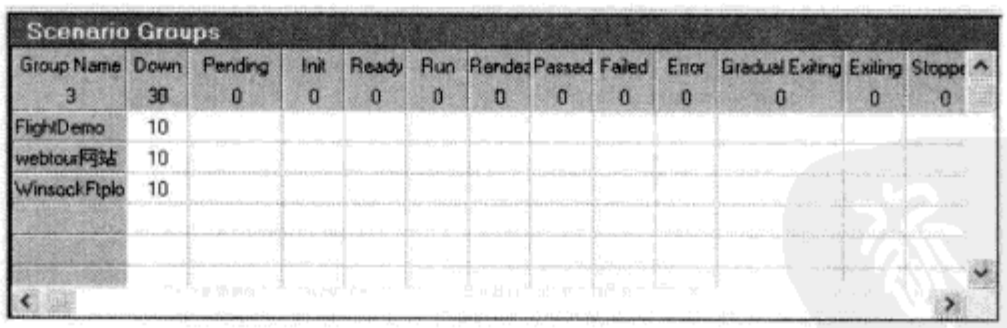


图 6-42　场景组中各个虚拟用户组的状态信息

在上面的场景图中，一共要运行三个虚拟用户脚本（虚拟用户组），每个虚拟用户组有 10 个 Vuser，目前这 30 个虚拟用户都处于初始的关闭状态（Down）。下面我们来看一下，这些状态都各自是什么含义（见表 6-1）。

表 6-1 虚拟用户组状态含义详解

状　态	含　义
关闭（Down）	Vuser 处于关闭状态
挂起（Pending）	Vuser 已经准备好，可以进行初始化，正在将脚本文件传输到负载生成器（Load Generator）
初始化（Init）	Vuser 正在初始化，执行对应的 vuser_init 函数
就绪（Ready）	Vuser 已经执行完初始化操作，可以运行 Action 了
正在运行（Run）	Vuser 脚本正在运行
集合点（Rendezvous）	Vuser 到达了集合点，正在等待条件满足，然后释放
完成并通过（Passed）	Vuser 已经运行结束，并且是成功通过
完成但失败（Failed）	Vuser 已经运行结束，并且是失败
错误（Error）	Vuser 发生了错误，要了解错误的完整说明，就要查看单个 Vuser 的详细状态日志
逐步退出（Gradula Exiting）	Vuser 正在运行退出前的最后一次迭代
退出（Exiting）	Vuser 已经完成操作，正在退出
停止（Stopped）	Vuser 被停止

其中 Down、Passed、Stopped、Error 是最终状态，其他的 Init、Ready、Rendezvous、Exiting 等是中间状态，也就是说，场景执行之前或执行结束后，所有的 Vuser 应该处于最终状态。如果虚拟用户长时间处于中间状态，那么就需要按照上一节提到的控制方法对其进行强制状态变更。

我们可以在“运行”视图右上角的框中查看正在运行的场景信息，如图 6-43 所示。

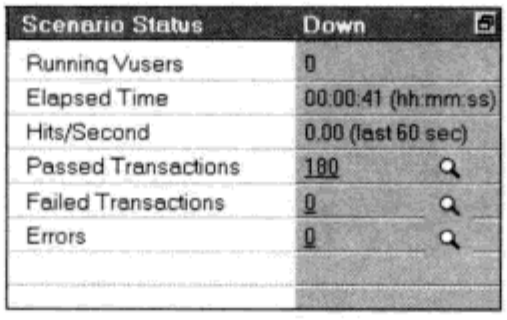


图 6-43 场景概要信息

各项场景状态的含义如表 6-2 所示。

表 6-2　场景状态信息详解

状　态	含　义
场景状态（Scenario Status）	说明目前场景是处在运行（Run）还是关闭（Down）状态
正在运行的 Vuser（Running Vusers）	说明现在有多少虚拟用户正在运行
已用时间（Elapsed Time）	从场景开始到现在一共运行了多长时间
每秒点击数（Hits/Second）	场景每秒对服务器发起的 HTTP 请求数，这是衡量一个场景负载大小的指标
通过的事务数（Passed Transactions）	场景运行到目前成功通过的事务数
失败的事务数（Failed Transactions）	场景运行到目前失败的事务数
错误数（Errors）	场景从运行到目前发生的错误数



提示　因为事务是我们在 VU 脚本中自己定义的。在脚本中被定义的任何事务都将统计并显示在场景概要信息框里，所以事务数的多少并不直接反映场景对服务器压力的大小。

如果我们需要查看各项状态的详细信息，可以直接点击此项，将会弹出详细信息对话框。

2. 事务详细信息

我们可以在“事务”对话框中查看各个事务的详细信息（见图 6-44）。要打开“事务”对话框，请单击“方案状态”窗口中“通过的事务数”或“失败的事务数”右侧的链接或放大镜按钮。

- 名称：列出脚本中各个事务的名称。
- TPS：显示每秒事务数。
- 已通过：列出已通过的事务的数量。
- 已失败：列出已失败的事务的数量。

- 已停止：列出已停止的事务的数量。

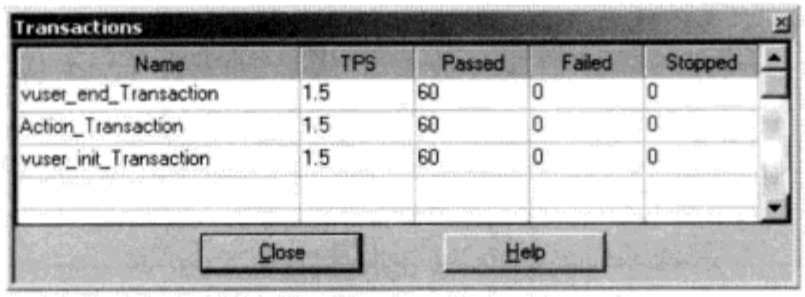


图 6-44 事务详细信息

3. 错误详细信息

单击错误列表右侧的“显示快照”按钮，将打开输出窗口，其中显示错误日志信息的列表如图 6-45 所示。

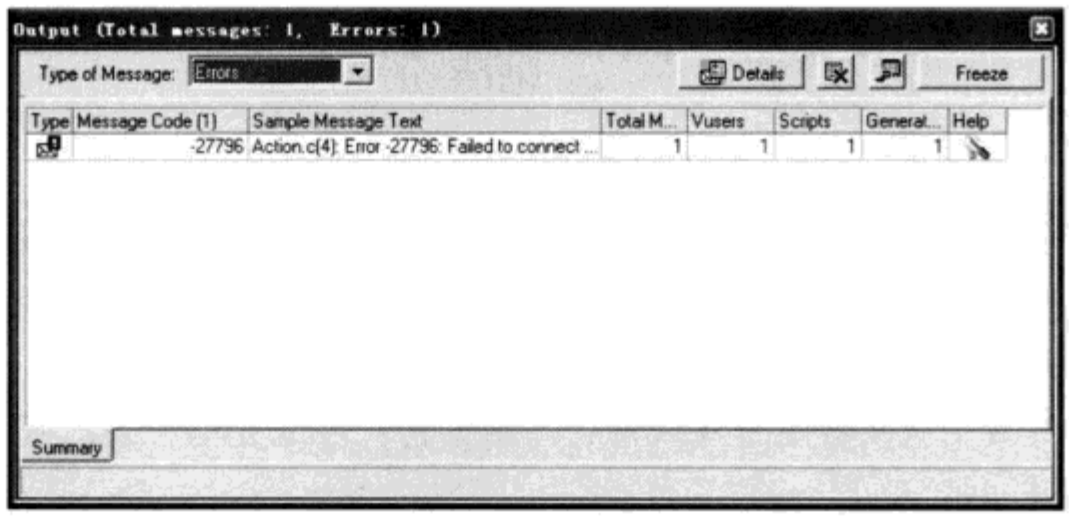


图 6-45 场景消息输出窗口

要查看消息的详细信息，请选中消息后单击“Details”按钮，将在输出窗口中打开“详细消息文本”框，其中显示完整的消息文本示例，如图 6-46 所示。

以上出错详细信息日志表明在访问 MI tours 演示网站首页时就连接超时，造成这个错误的原因可能是 Run-time Settings 浏览器的代理没有设置正确。

4. 按照不同模式查看错误

一个错误消息可能与多个虚拟用户、多个脚本、多个负载生成器（Load

Generator）相关，如果我们要查看更加详细的信息，那么可以点击相应列的蓝色链接。

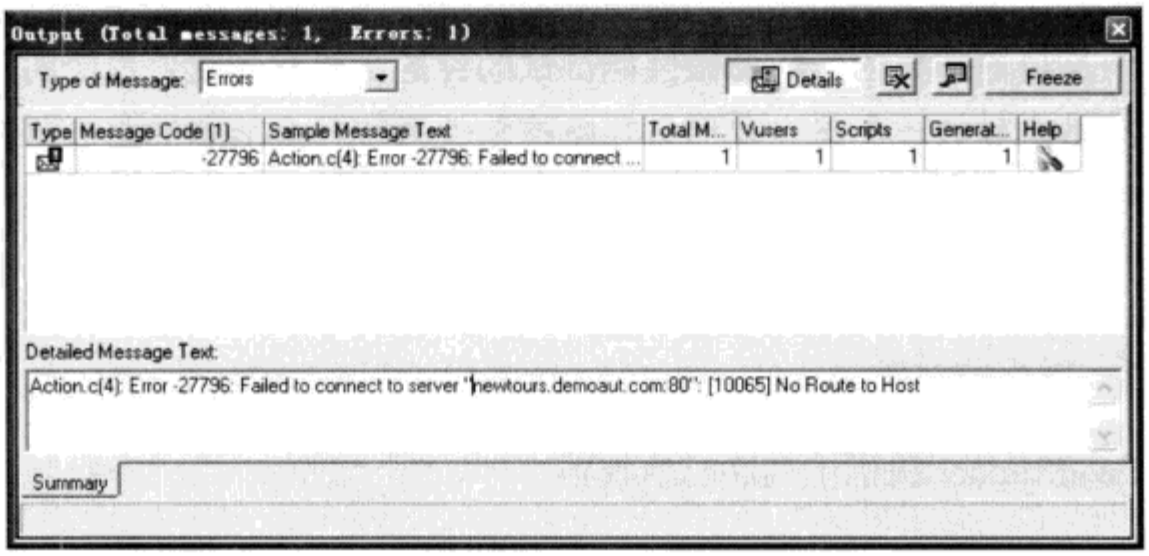


图 6-46　查看出错详细信息

比如，我们想要查看这个出错消息的详细信息，以确定脚本语句的出错位置，点击“Message”链接，可看到如图 6-47 所示的对话框。

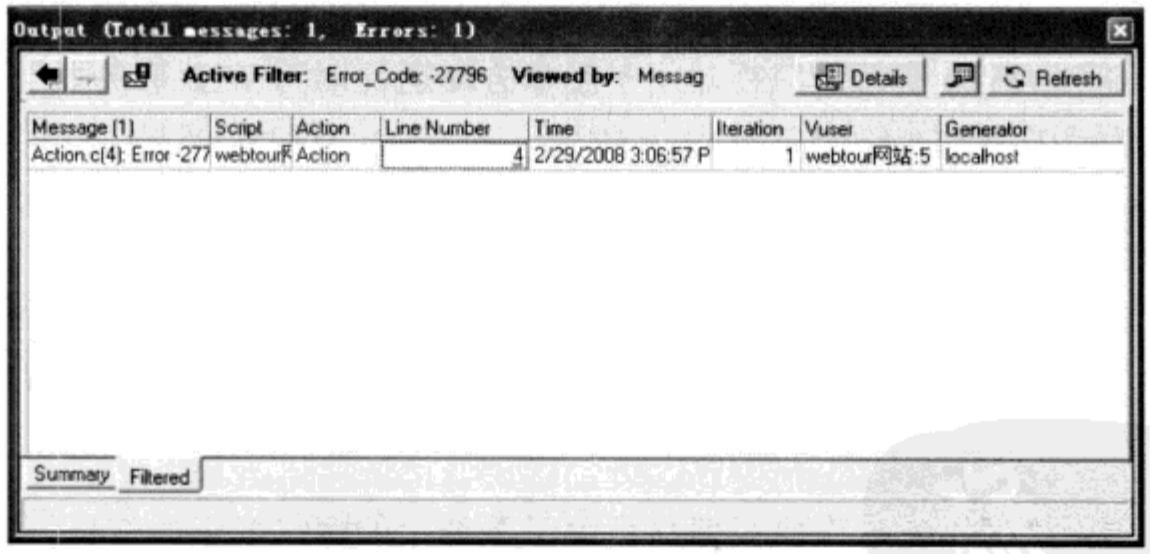


图 6-47　查看错误消息的详细信息

从以上对话框中我们可以看到，错误出现在名为 webtour 的脚本里，在执行第一次迭代时，运行到脚本第 4 行就出错了。点击 Script 蓝色链接，可以进一步打开脚本，并查看相关语句。

6.3.3 监视场景

查看场景状态信息可以知道哪个虚拟用户执行脚本的哪条语句出了错，但这远远不是我们想要的，我们做性能测试的目的是要找到软件系统的瓶颈，光凭客户端的错误显然还不够，我们必须监控软件系统各个节点的服务器的性能表现，以判断瓶颈到底出现在哪里。所以在本节我们要熟悉 LoadRunner 提供的监视器和如何去添加监视器并分析监视曲线图。

1．联机监视器

LoadRunner 提供下列联机监视器：

（1）“运行时”监视器显示参与方案的 Vuser 的数目和状态，以及 Vuser 所生成的错误数量和类型。此外还提供用户定义的数据点图，其中显示 Vuser 脚本中的用户定义点的实时值。

（2）“事务”监视器显示方案执行期间的事务速率和响应时间。

（3）“Web 资源”监视器用于度量方案运行期间 Web 服务器上的统计信息。它提供关于方案运行期间的 Web 连接、吞吐量、HTTP 响应、服务器重试和下载页的数据。

（4）“系统资源”监视器测量方案运行期间使用的 Windows、UNIX、TUXEDO、SNMP 和 Antara FlameThrower 资源。要激活系统资源监视器，必须在运行方案之前设置监视器选项。

（5）“网络延迟”监视器显示关于系统上的网络延迟的信息。要激活网络延迟监视器，必须在运行方案之前设置要监视的网络路径。

（6）“防火墙”监视器用于度量方案运行期间防火墙服务器上的统计信息。要激活防火墙监视器，必须在运行方案之前设置要监视的资源列表。

（7）“Web 服务器资源”监视器用于度量方案运行期间 Apache、Microsoft IIS、iPlanet（SNMP）和 iPlanet/Netscape Web 服务器上的统计信息。要激活 Web 服务

器资源监视器，必须在运行方案之前设置要监视的资源列表。

（8）“Web 应用程序服务器资源”监视器用于度量方案运行期间 Web 应用程序服务器上的统计信息。要激活 Web 应用程序服务器资源监视器，必须在运行方案之前设置要监视的资源列表

（9）“数据库服务器资源”监视器用于度量与 SQL Server、Oracle、Sybase 和 DB2 数据库有关的统计信息。要激活数据库服务器资源监视器，必须在运行方案之前设置要监视的度量列表。

（10）“流媒体”监视器用于度量 Windows Media 服务器、RealPlayer 音频/视频服务器及 RealPlayer 客户端上的统计信息。要激活流媒体监视器，必须在运行方案之前设置要监视的资源列表。

（11）“ERP/CRM 服务器资源”监视器用于度量方案运行期间 SAP R/3 系统服务器、SAP Portal、Siebel Web 服务器和 Siebel Server Manager 服务器的统计信息。要激活 ERP/CRM 服务器资源监视器，必须在运行方案之前设置要监视的资源列表。

（12）“Java 性能”监视器用于度量 Java 2 Platform, Enterprise Edition（J2EE）对象及使用 J2EE 和 EJB 服务器计算机的 Enterprise Java Bean（EJB）对象的统计信息。要激活 Java 性能监视器，必须在运行方案之前设置要监视的资源列表。

（13）“应用程序部署解决方案”监视器用于度量方案运行期间 Citrix MetaFrame XP 和 1.8 服务器的统计信息。要激活应用程序部署解决方案监视器，必须在运行方案之前设置监视器选项。

（14）“中间件性能”监视器用于度量方案运行期间 TUXEDO 和 IBM WebSphere MQ 服务器上的统计信息。要激活中间件性能监视器，必须在运行方案之前设置要监视的资源列表。

所有的监视器都可用于查看总结方案时所收集的数据摘要。各个监视器的介绍可参看 LoadRunner 的 Controller 手册。下面以系统资源类型中的 Windows 资源

监视器为例，介绍怎样添加。

2. 在 Controller 中启动监控器

在运行过程中，可以监视各个服务器的运行情况（DataBase Server、Web Server 等）。

监视场景通过添加性能计数器来实现。

在“Available Graphs”中的树形栏中列出了 LoadRunner 支持的监视器类型，我们以 Windows Resources 为例，介绍在 Controller 中怎么启动监视器。

由于 Windows 2000 Professional、Server 以及 Advanced Server 提供的计数器不完全相同，这里我们讨论将以 Server 为基准。监视场景需要在 Run 视图中设置，如图 6-48 所示。

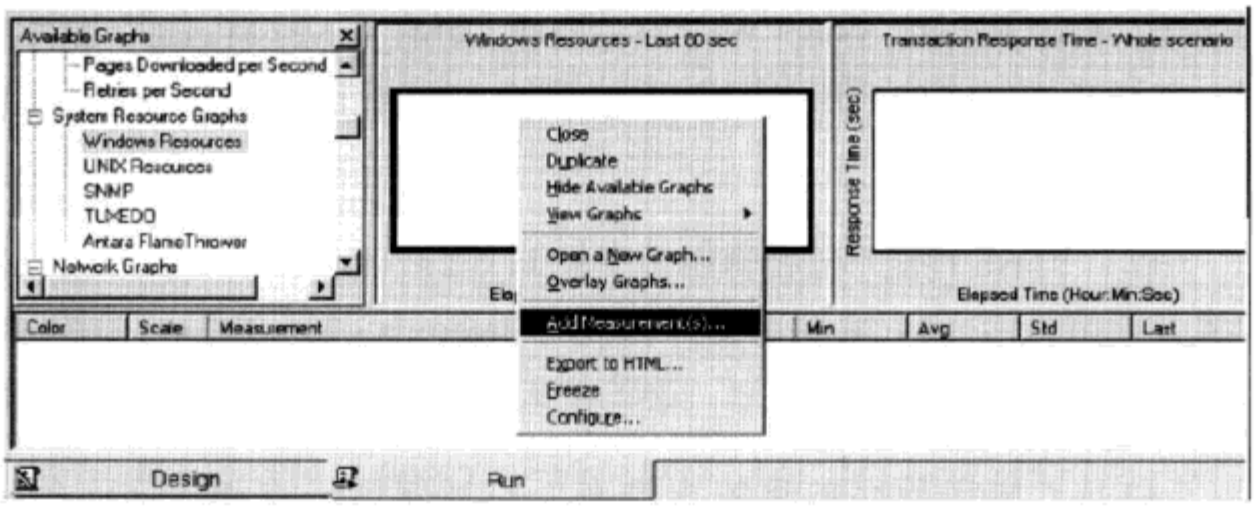


图 6-48　在监视器中添加 Windows Resources

然后，出现添加计数器的对话框，如图 6-49 所示。

可以选择添加多个计数器，添加成功后，会在监视图中显示各个计数器值随着时间变化的曲线。

3. 配置监视器

创建监视器后，可以配置监视器，使之“个性化”。

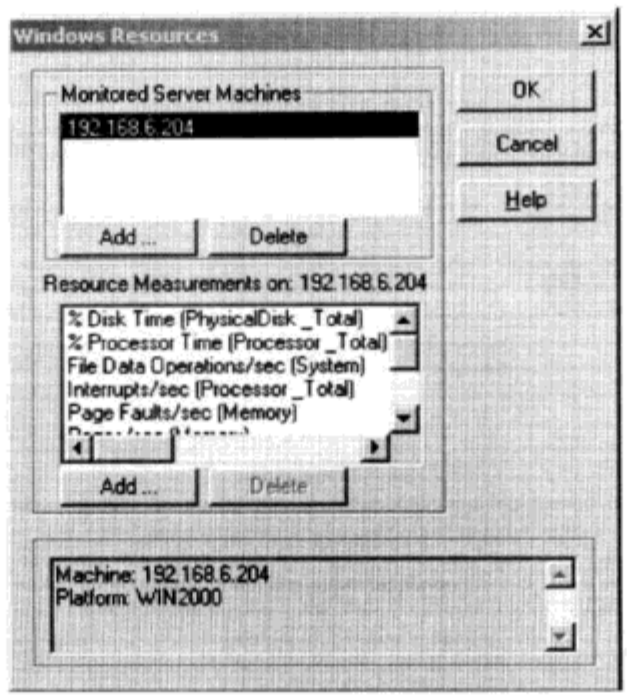


图 6-49　添加监视指标

通过 LoadRunner 可以配置联机监视器的设置。可以设置图的度量和属性，例如采样时间、线条颜色及图的比例。

（1）监视器选项

在“Tools”>“Options”中的“Monitors”选项卡中进行全局采样速率、错误处理、调试和频率设置，如图 6-50 所示。

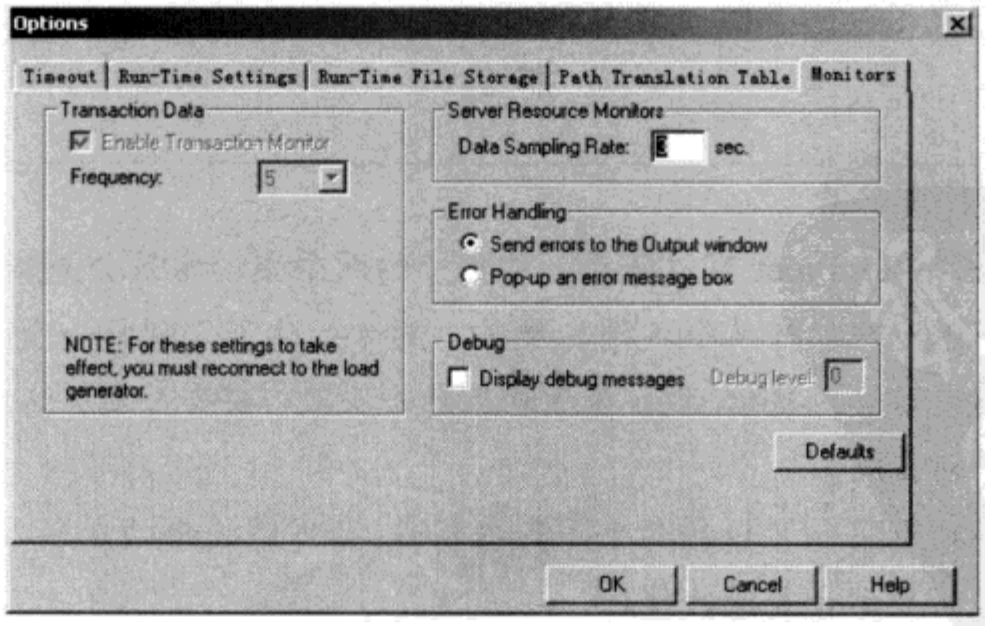


图 6-50　设定监视器采样频率、出错处理等

（2）图属性

选择“监视器”>“联机图”>“配置”，或者右键单击某个图并选择“配置”，将打开“图配置”对话框，如图 6-51 所示，其中显示刷新率、显示类型、*X* 轴（图的时间）以及 *Y* 轴比例。

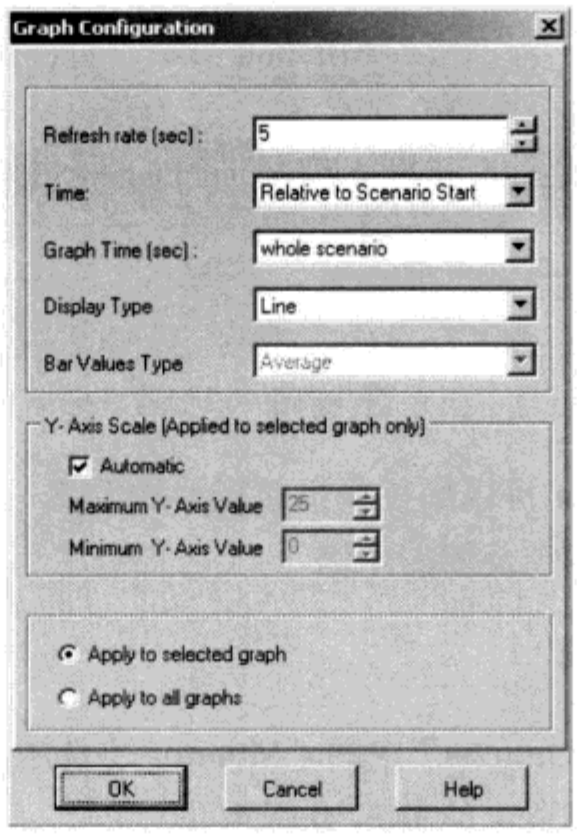


图 6-51　设定监视器的显示方式

提示　图的合并

通过 LoadRunner 可以将同一方案中的两个图的结果合并到一个图中。通过合并，可以一次比较几个不同的度量。例如，你可以制作一个合并图，以已用时间的函数的形式显示 Web 吞吐量和点击次数。要合并图，这些图的 *X* 轴的度量单位必须相同。

叠加共用同一 *X* 轴的两个图的内容时，合并图左侧的 *Y* 轴显示当前图的值，右侧的 *Y* 轴显示合并图的值。



保存方案时，将同时保存联机监视器配置设置。

6.3.4　高级——用户自定义数据采集点

如果我们想采集的数据不在 LoadRunner 监视器之列的话，那就需要用户自定义数据点了。通过在脚本适当位置插入自定义数据采集函数，这样每当脚本运行到此处，就采集数据一次。在测试场景运行结束后，LoadRunner 会生成图表，和其他监视器数据一样，反映被采集数据随时间变化的趋势。

这个函数就是 **lr_user_data_point** 函数（对于 GUI Vuser 为 **user_data_point**，对于 JavaVuser 为 **lr.user_data_point**）。lr_user_data_point 函数有两个入口参数，第一个是数据点的名字，第二个是数据值。例如，在下面这个 Oracle 脚本中，我们使用 SQL 语句得到 Oracle 某个 instance 的 v$sysstat 数据，然后把它写在自己定义的数据采集点中，以便分析。与传统的监视器相比，这样的自定义数据点的优势在于可以同时采集多个 Oracle instance 的数据。

```
/*
Atof 是一个 C 函数，其作用是将一个字符串类型转换为浮点类型
*/
double atof(const char *s);

Actions()
{
double val1;
double val2;
double val3;
double val4;
double val5;
double val6;
  /*
   * 当前并发用户
   */
 lrd_ora8_handle_alloc(OraEnv1, STMT, &OraStm3, 0);
 lrd_ora8_stmt(OraStm3,  "select  value  from  v$sysstat  where
```

```
statistic# = 1\n", 1, 0, 0);
    lrd_ora8_exec(OraSvc1, OraStm3, 0, 0, &uliRowsProcessed, 0, 0, 0,
0, 0);
    lrd_ora8_bind_col(OraStm3, &OraDef2, 1, &VALUE_D17, 0, 0);

    lrd_ora8_save_col(OraStm3,1,1,0,"Saved_VALUE_D17");
    lrd_ora8_fetch(OraStm3, -1, 15, &uliFetchedRows, PrintRow6, 2, 0,
0);
       GRID8(6);
    lrd_handle_free(&OraStm3, 0);

       /*
        * 当前打开的游标
        */
       lrd_ora8_handle_alloc(OraEnv1, STMT, &OraStm4, 0);
       lrd_ora8_stmt(OraStm4, "select value from v$sysstat where
statistic# =3\n", 1, 0, 0);
       lrd_ora8_exec(OraSvc1, OraStm4, 0, 0, &uliRowsProcessed, 0, 0,
0, 0, 0);
       lrd_ora8_bind_col(OraStm4, &OraDef3, 1, &VALUE_D18, 0, 0);

    lrd_ora8_save_col(OraStm4,1,1,0,"Saved_VALUE_D18");
       lrd_ora8_fetch(OraStm4, -1, 15, &uliFetchedRows, PrintRow7, 2,
0, 0);
       GRID8(7);
       lrd_handle_free(&OraStm4, 0);


    /*转换为 double 型*/
    val1=atof(lr_eval_string("<Saved_VALUE_D17>"));
    val2=atof(lr_eval_string("<Saved_VALUE_D18>"));
    /*定义两个数据点*/
    lr_user_data_point ("CurrentLogons",val1);
    lr_user_data_point ("CurrentCursors",val2);

    /*
       注意:在本脚本中没有 Think Time。因此在 Controller 运行时可以在 Run-time
Settings 中设置两次 Iteration 的间隔为固定的时间，比如 10 秒，这样，脚本就能大
概 10 秒钟被重复一次，按 10 秒的频率去采集数据了
    */
```

```
    return VUSER_OK;
}
```

有关 lr_user_data_point 函数的详细使用方法可参看 LoadRunner 的函数手册。

在 Analysis 中就会生成如图 6-52 所示的图表。

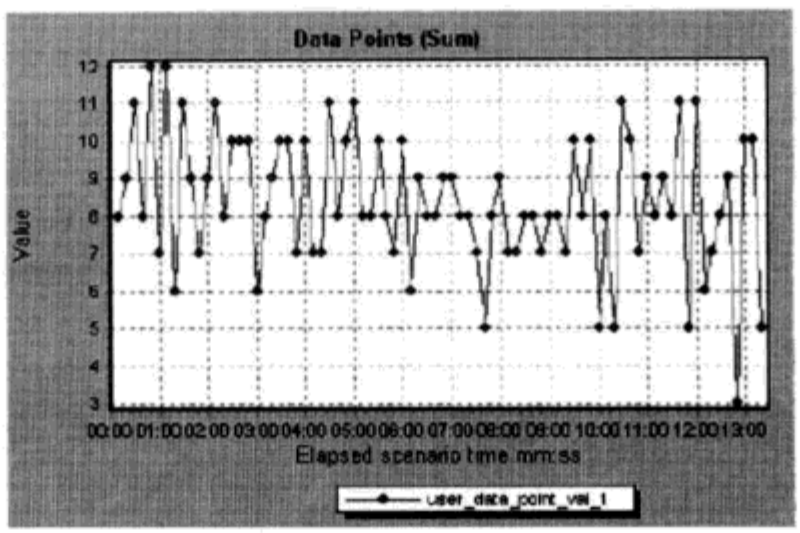


图 6-52　用户自定义数据点生成的监视器曲线图

6.3.5　主要的计数器

内存是第一个监视对象，确定系统瓶颈的第一个步骤就是排除内存问题。内存短缺可能会引起各种各样的问题。内存计数器一览见表 6-3。

内存问题主要检查应用程序是否存在内存泄漏。如果发生了内存泄漏，Process/Private Bytes 计数器和 Process/Working Set 计数器的值往往会升高，同时 Available Bytes 的值会降低。

内存泄漏应该通过一个长时间的，用来研究分析当所有内存都耗尽时，应用程序反应情况的测试来检验。

表 6-3　内存计数器一览

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
Memory	Available MBytes	物理内存的可用数（单位为 MB）。默认情况下 IIS 5.0 使用 50%的可用物理内存，作为 IIS 的文件缓存（Lile Cache）。IIS 基本占用 2.5MB，每个附加连接将在此基础上占用 10KB 左右	至少要有 10%的物理内存值

续表

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
Memory	Page/sec Page Faults/sec Pages Input/sec Page Reads/sec Transition Faults/sec	当处理器向内存指定的位置请求一页（可能是数据或代码）出现错误时，这就构成一个 Page Fault。如果该页在内存的其他位置，该错误被称为软错误（用 Transition Fault/sec 计数器衡量）；如果该页必须从硬盘上重新读取时，被称为硬错误。许多处理器可以在有大量软错误的情况下继续操作。但是，硬错误可以导致明显的拖延。Page Faults/sec 是处理器每秒钟处理的错误页（包括软错误和硬错误） Pages Input/sec 是为了解决硬错误页，从硬盘上读取的页数，而 Page Reads/sec 是为了解决硬错误，从硬盘读取的次数。如果 Page Reads/sec 比率持续保持为 5，表示可能内存不足 Pages/sec 是指为解析硬错误页从磁盘读取或写入磁盘的页数	Page/sec 推荐值为 00~ 20（如果服务器没有足够的内存处理其工作负荷，此数值将一直很高。如果大于 80，表示有问题） 这些计数器的值比较低，说明 Web 服务器响应请求比较快，否则可能是服务器系统内存短缺引起（也可能是缓存太大，导致系统内存太少）。Page Input/sec 的值可以衡量出硬错误页发生的速率，通常它的值会大于或者等于 Page Reads/sec
Memory	Cache Bytes	文件系统缓存（File System Cache），默认情况下为 50%的可用物理内存。如 IIS 5.0 运行内存不够时，它会自动整理缓存。需要关注该计数器的趋势变化	默认情况下为 50%的可用物理内存
Internet Information Scervices Global	File Cache Hits% File Cache Flushes File Cache Hits	File Cache Hits %是文件缓存命中全部缓存需求的比例，反映了 IIS 的文件缓存设置的工作情况。而 File Cache Hits 是文件缓存命中的具体值，File Cache Flushes 是自服务器启动之后文件缓存刷新次数，如果刷新太慢，会浪费内存；如果刷新太快，缓存中的对象会太频繁的丢弃生成，起不到缓存的作用。通过 File Cache Hits 和 File Cache Flushes 可以得到一个适当的刷新值（参考 IIS 的设置 ObjectTTL 、MemCacheSize 、MaxCacheFileSize）	（对于一个大部分是静态网页组成的网站）File Cache Hits%在 80%左右属于非常好

续表

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
Memory	Pool Paged Bytes Pool Nonpaged Bytes	这两个计数器监视服务器上各个进程的分页池字节数和非分页池字节数	在访问数比较固定的情况下，Pool Nonpaged Bytes 是比较固定的，如果访问数逐步增加，该值会缓慢的增加
Process	Virtual Bytes（实例 inetinfo 、 dllhost ） Working Set（实例 inetinfo 、 dllhost ） Dllhost#n 进程都要添加计数器	Virtual Bytes 计数器监视 IIS 5.0 保留的虚地址空间的数量，实例化为 inetinfo 进程（IIS 运行的核心）和 Dllhost 进程（隔离/连接池的应用程序必需的）。Working Set 计数器反映了每个进程使用的内存页的数量。系统的内存页（Pool Page）只能由操作系统的核心模块直接访问，用户进程不能访问。运行 IIS 5.0 的服务器上，负责 Web 连接的线程以及它需要的一些对象都保存在未分页的池中（Nonpaged Pool），比如文件句柄和 socker 连接	
Process	Private Bytes	指这个处理不能与其他处理共享的、已分配的当前字节数	
Memory	Committed Bytes	Committed Byte 是指以字节表示的确认虚拟内存。（确认内存是指为磁盘分页文件在磁盘上保留的空间，以便在需要将其写回磁盘时使用）	推荐不超过物理内存的 75%

1. Processor 相关

Processor 相关计数器一览如表 6-4 所示。

表 6-4　Process 相关计数器一览

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
Sytem	Processor Queue Length	Processor Queue Length 是指处理队列中的线程数。即使在有多个处理器的计算机上处理器时间也会有一个单队列。不像磁盘计数器，这个计数器仅计数就绪的线程，而不计数运行中的线程。如果处理器队列中总是有两个以上的线程，通常表示处理器堵塞	小于 2。显示在由 Web 服务器所有处理器共享的队列中等待执行的线程数。处理器瓶颈会导致该值持续大于 2

续表

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
Processor	%Processor Time	CPU 使用率。这是查看处理器饱和状况的最佳计数器。显示所有 CPU 的线程处理时间。如果一个或多个处理器的该数值持续超过 90%，则表示此测试的负载对于目前的硬件过于沉重。为多处理器服务器添加该计数器的 0~*x* 个实例	小于 75%。排除内存因素，如果该计数器的值比较大，而同时网卡和硬盘的值比较低，那么可以确定 CPU 瓶颈
System	Context Switches/sec	Context Switches/sec 指计算机上的所有处理器全都从一个线程转换到另一个线程的综合速率。当正在运行的线程自动放弃处理器时出现上下文转换，由一个有更高优先就绪的线程占先或在用户模式和特权（内核）模式之间转换以使用执行或分系统服务。它是在计算机上的所有处理器上运行的所有线程的 Thread：Context Switches/sec 的总数并且用转换数量衡量。在系统和线程对象上有上下文转换计数器	如果切换次数到 5000*CPU 个数和 10000*CPU 个数时，说明它忙于切换线程而不是处理 ASP 脚本
Processor	%Privileged Time	%Privileged Time 是在特权模式下处理线程执行代码所花时间的百分比。当调用 Windows 系统服务时，此服务经常在特权模式下运行，以便获取对系统专有数据的访问。在用户模式下执行的线程无法访问这些数据。对系统的调用可以是直接的（explicit）或间接的（implicit），例如页面错误或中断。不像某些早期的操作系统，Windows 除了使用用户和特权模式的传统保护模式之外，还使用处理边界作为分系统保护。某些由 Windows 为你的应用程序所做的操作除了出现在处理的特权时间内，还可能在其他子系统处理中出现	
Thread	Context Switches/sec（实例化 inetinfo 和 dllhost 进程）	如果你决定要增加线程字节池的大小，就应该监视这三个计数器（包括上面的一个）。增加线程数可能会增加上下文切换次数，这样性能不会上升反而会下降。如果十个实例的上下文切换值非常高，就应该减小线程字节池的大小	

续表

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
Processor	Interrupts/sec　%DPC Time	这两个计数器能够反映处理器用在处理中断以及推迟处理调用的时间。 Interrupts/sec 指处理器每秒钟接收并维护的硬件中断的平均值。正常的线程操作在中断时悬停。大多数的系统时钟每隔 10 毫秒中断处理器一次，形成了间隔活动的后台	如果处理器使用率超过 90%且%Interrupt Time 大于 15%，则处理器可能负荷过重，并发生中断

判断应用程序是否存在处理器瓶颈的方法：如果 Processor Queue Length 显示的队列长度保持不变（≥2），并且处理器的利用率%Processor Time 超过 90%，那么很有可能存在处理器瓶颈。

如果发现 Processor Queue Length 显示的队列长度超过 2，而处理器的利用率却一直很低，那么或许更应该去解决处理器阻塞问题，这里处理器一般不是瓶颈。

如果系统由于应用程序代码效率低下或者系统结构设计有缺陷而导致大量的上下文切换（Context Switches/sec 显示的上下文切换次数比较大），那么就会占用大量的系统资源。

如果系统的吞吐量降低并且 CPU 的使用率很高，并且此现象发生时切换水平在 15000 以上，那么意味着上下文切换次数过高。同时还可以比较 Context Switches/sec 和%Privileged Time 来判断上下文切换是否过量。如果后者的值超过 40%，且上下文切换的速率也很高，那么应该检查为什么会产生这样高的上下文切换。

2. 网络吞吐量以及带宽

网络相关计数器一览如表 6-5 所示。

表 6-5 网络相关计数器一览

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
Network Interface	Bytes Total/sec	Bytes Total/sec 为发送和接收字节的速率，包括帧字符在内。判断网络连接速率是否是瓶颈，可以用该计数器的值和目前网络的带宽比较	该计数器的值和目前网络的带宽相除，结果应该小于 50%
Web Service	Maximum Connections 、Total Connection Attempts	Maximum Connections（最大连接数）：是和 Web 服务同时建立起的最大连接数 Total Connection Attempts（连接尝试总数）：是从服务启动时利用 Web 服务尝试连接的总数。该计数器应用于全部所列的实例	

3. 磁盘相关

磁盘相关计数器一览如表 6-6 所示。

表 6-6 磁盘相关计数器一览

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
Processor	%Processor Time %Privileged Time	CPU 使用率 该计数器对应于处理器执行 Windows 2000 内核命令（如处理 SQL Server I/O 请求）所用时间的百分比。如果 Physical Disk 计数器的值很高时该计数器的值也一直很高，则考虑使用速度更快或效率更高的磁盘子系统	
PhysicalDisk	%Disk Time	%Disk Time 指所选磁盘驱动器忙于为读或写入请求提供服务所用的时间的百分比。在记录该计数器之前，请在 Windows 2000 的命令行窗口中运行 diskperf-yD。若数值持续超过 80%，则可能是内存泄露	
PhysicalDisk	Average Disk Queue Length	指读取和写入请求（为所选磁盘在实例间隔中队列）的平均数	
PhysicalDisk	Average Disk Read Queue Length	指读取请求（为所选磁盘在实例间隔中队列）的平均数	

续表

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
PhysicalDisk	Average Disk Write Queue Length	指写入请求（为所选磁盘在实例间隔中队列）的平均数	
PhysicalDisk	Average Disk sec/Read	指以秒计算的在此盘上读取数据的所需平均时间	
PhysicalDisk	Average Disk sec/Transfer	指以秒计算的在此盘上写入数据的所需平均时间	
PhysicalDisk	Disk Reaks/sec	指在此盘上读取操作的速率	
PhysicalDisk	Disk Writes/sec	指在此盘上写入操作的速率	

判断磁盘瓶颈的方法是通过以下公式来计算：

每磁盘的 I/O 数=[读次数+(4×写次数)]/磁盘个数

如果计算出的每磁盘的 I/O 数大于磁盘的处理能力，那么磁盘存在瓶颈。

4．Web 应用程序

这里以 ASP.NET 开发的 Web 应用程序为例进行说明。Web 应用程序计数器一览如表 6-7 所示。

表 6-7　Web 应用程序计数器一览

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
ASP.NET Applications	Request/Sec Request Executing	每秒执行的请求数 当前执行的请求数	如果 Request/Sec 的值比较小，则 Web 程序可能是瓶颈
ASP.NET	Request Wait Time Request Executing Time Request Queued	最近的请求在队列中等待的毫秒数 执行最近的请求所用的毫秒数 等候处理的请求数。该计数器应保持接近 0。超过 IIS 队列长度会出现“服务太忙”错误	Request Wait Time 和 Request Queued 在理想状况下应该接近 0，如果这两个值太大，那么需要重写代码提高性能

5．IIS 5.0

IIS 计数器一览如表 6-8 所示。

表 6-8 IIS 计数器一览

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
Web Service	Bytes Sent/Sec	“送出字节数/秒”是 Web 服务送出数据字节的比率	
Web Service	Bytes Received/Sec	“接收字节数/秒”是 Web 服务接收数据字节的比率	
Web Service	Get Request/Sec	使用 Get 方法的 HTTP 请求速率。Get 请求用于基本文件获取或图像地图，它们可和表格一起使用	
Web Service	Rost Request/Sec	使用 Post 方法的 HTTP 请求速率。Post 请求用于表格或网关请求	

6. SQL Server

这里针对 SQL Server 2000，而且只是列出比较关键的几个。更加详细的信息可以参考 SQL Server 的联机文档。

SQL Server 常用计数器一览如表 6-9 所示。

表 6-9 SQL Server 常用计数器一览

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
Processor	%Processor Time	CPU 使用率	
SQL Server:General Statistics	Logins/sec	这是每秒登录到 SQL Server 的计数	
SQL Server:Cache Manager	Cache Hit Ratio (all instances)	显示在高速缓存中找到数据的命中率。如果数值持续小于 85%，则表示内存有问题	
SQL Server:General Statistics	User Connections	显示当前 SQL 用户数。与 Active Server Pages:Requests/sec 计数器进行比较，可帮助了解脚本对 SQL Server 的影响程度。如果差别过大，则表示测试脚本不能有效地对 SQL Server 进行应力测试	
SQL Server:Locks	Lock Waits/sec	显示在当前进程完成之前强制其他进程等待的每秒锁定请求的数量。如果该值始终大于 0，则表示事务有问题	
SQL Server:Buffer Manager	Buffer Cache Hit Ratio	计数器值依应用程序而定，但比率最好为 90%或更高。增加内存直到这一数值持续高于 90%，表示 90%以上的数据请求可以从数据缓冲区中获得所需数据	

续表

Object（对象）	Counters（计数器名称）	Description（描述）	参考值
SQL Server:SQL Statistics	Batch Requests/sec	每秒收到的 Transact-SQL 命令批数。这一统计信息受所有约束（如 I/O、用户数、高速缓存大小、请求的复杂程度等）影响。批请求数值高意味着吞吐量很好	
SQL Server:Buffer Manager	Lazy Writes/sec	每秒被缓冲区管理器的惰性写入器写入的缓冲区数。惰性写入器是一个系统进程，其主要任务是刷新成批的老化的脏缓冲区（指包含更改的缓冲区，这些更改必须写回磁盘，才能使该缓冲区由其他页重新使用），并使之可由用户进程使用。惰性写入器消除了为创建可用缓冲区而频繁执行检查点的需要	
SQL Server:Buffer Manager	Page Reads/sec	每秒发出的物理数据库页读取数。这一统计信息显示的是在所有数据库间的物理页读取总数。由于物理 I/O 的开销大，可以通过使用更大的数据高速缓存、智能索引、更高效的查询或者改变数据库设计等方法，使开销减到最小	
SQL Server:Databases	Transactions/sec	每秒为数据库启动的事务数	

注 意　**网络延迟（Network Delay）**

如果要监视的两台计算机在同一个局域网络内，建议不要使用 Network Delay Monitor。因为在同一局域网内，Network Delay 会非常的小，网络监视器会有足够的时间在每秒钟内发送成百上千的请求，这样会导致源计算机（Source Machine）的 CPU 和内存超负荷工作。

在默认情况下，“Enable display of network nodes by DNS names”选择是没有选中的，因为选中它会明显地降低该监视器的速度。

6.3.6　实例——分析实时监视图表

从 Controller 的监控区域，我们能够对被测系统的状态有个大概的了解。

1. 实例 1：事务响应时间是否在可接受的时间内？哪个事务用的时间最长？

看 Transaction Response Time 图，可以判断完成每个事务所用的时间，从而可以判断出哪个事务用的时间最长，哪些事务用的时间超出预定的可接受时间。从图 6-53 可以看出，随着用户数的不断增加，Login 事务的响应时间增长得最快！

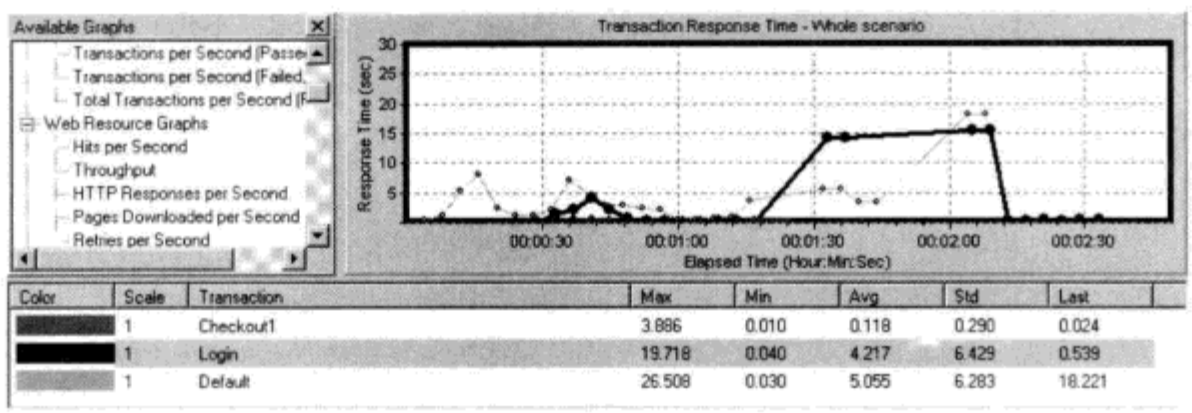


图 6-53 监视器事务响应时间图表

2. 实例 2：网络带宽是否足够？

如图 6-54 所示，“Throughput”图显示在场景运行期间的每一秒钟，从 Web Server 上接收到的数据量的值。拿这个值和网络带宽比较，可以确定目前的网络带宽是否是瓶颈。

如果该图的曲线随着用户数的增加，没有随着增加，而是呈比较平的直线，说明目前的网络速度不能够满足目前的系统流量。

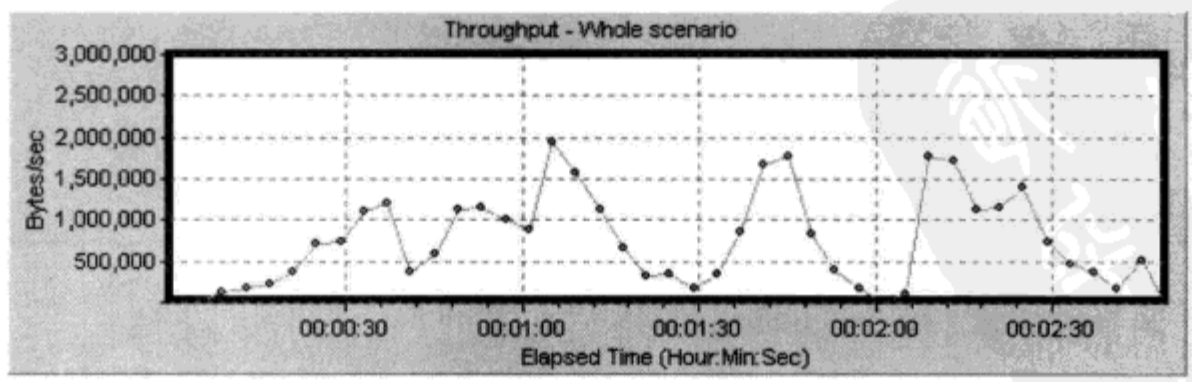


图 6-54 监视器网络吞吐量图

3. 实例 3：硬件和操作系统能否处理高负载？

如图 6-55 所示，“Windows Resources”图实时地显示了 Web Server 系统资源的使用情况。利用该图提供的数据，可以把瓶颈定位到特定机器的某个部件。

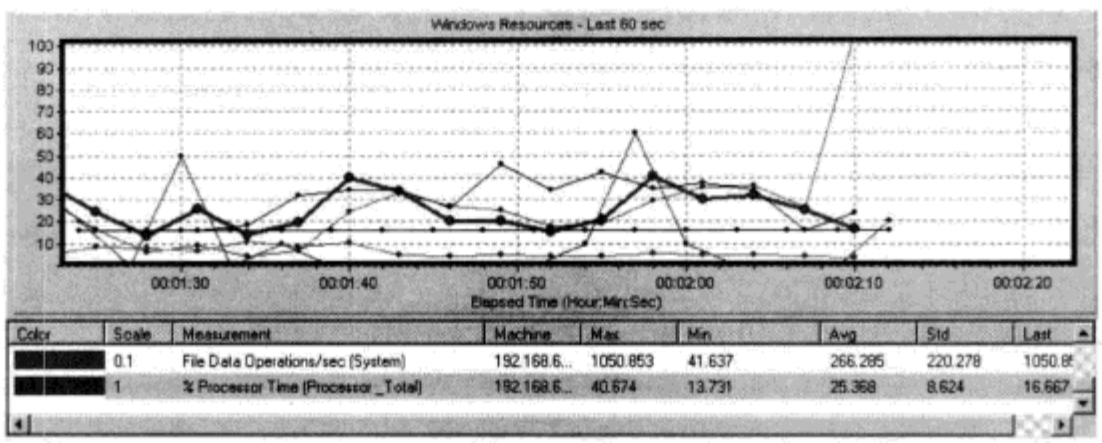


图 6-55　监视器的 Windows 资源指标监视曲线

6.4　场景运行后

场景成功运行后，可手工或自动设置开启 Analysis 对结果数据进行分析。



应对运行的场景进行保存，以便回归测试中再次使用。

6.5　高级——如何合理规划性能测试拓扑

假设通过性能测试需求分析，我们需要创建一个性能测试场景，并发 500 个 Web 虚拟用户，这时，我们需要考虑：

（1）选用什么样软硬件配置的机器作为测试机？

（2）500 个并发用户需要几台测试机才能够用？

在性能测试执行之前，一定要把上面的问题搞清楚，主要是为了避免将来性能测试执行时瓶颈出现在客户端，客户端承载了太多的压力，而没有真正地提交

到服务器端上去。这种情况下，我们会能看到客户端 CPU 占用率高居不下，响应速度十分缓慢，甚至出现宕机的情形。

实际上，针对特定的性能测试需求，建立多大规模的测试机群才算合理，与多个因素有关，包括：测试机的软硬件配置、测试机的数量、脚本的复杂程度、网络的情况等。

HP 为用户提供了一份 loadrunner 在不同环境下运行参数的 footprint 文档，不同版本的 LoadRunenr 有不同的 footprint，用户可以在 HP 的官方网站进行下载。

下面我们参考 lr8.1 footprint 文档，进行拓扑的规划。

Protocol	Win2000, 1 CPU, 1GB RAM			WinXP, 1 CPU, 1GB RAM			Win2003, 2 CPU, 2GB RAM		
	CPU, %	Process, MB	Thread, MB	CPU, %	Process, MB	Thread, MB	CPU, %	Process, MB	Thread, MB
WEB (Http/Html)	2.5	4.9	0.82	2.2	4.6	0.82	2.1	4.6	0.82
CITRIX	0.01	5.7	0.84	0.03	7	0.97	0	5.4	0.81
SAP GUI	39	43.3	3.77	46.4	32.2	4.1	49.8	33.6	4.14
ORACLE	6.61	11.5	0.62	6.25	11.4	0.58	4.11	11.3	0.59
ORACLE	6.74	11.5	0.68	7.73	11.4	0.68	6.9	11.3	0.67
ORACLE (NCA/WEB)	4.9	6.1	0.78	4.64	6.5	0.76	4.6	6.4	0.76
Web Services	6.58	35.7	1.39	1.69	35.1	1.37	2.4	35.4	1.38
Web Services	1.05	7.8	0.62	0.38	7.6	0.62	0.92	7.6	0.61
RTE		4.9	1		4.2	1			
Java/RMI	0.75	25.3	0.76	0.5	24.6	0.77	0.52	25	0.77
Winsock	0.04	3.1	0.32	0.01	2.9	0.32	0.02	2.8	0.32
POP3	1.04	3.3	0.33	0.97	3.1	0.32	0.99	3.1	0.32
FTP	12.3	3.4	0.38	6.9	3.1	0.37	10.5	3.1	0.37
COM/DCOM	7.2	3.5	0.31	6.58	3.3	0.31	5.6	3.3	0.31
MSSQL	14.1	3.2	0.32	15	3	0.32	14.9	3	0.32
TUXEDO	0	4.1		0	3.6		0	3.6	
Microsoft .Net	0.51	14	0.61	0.56	12.2	0.61	0.37	12.5	0.62
DB2	1.94	5.6	0.77	2.21	5.4	0.78	0.05	5.4	0.76

续表

Protocol	Win2000, 1 CPU, 1GB RAM			WinXP, 1 CPU, 1GB RAM			Win2003, 2 CPU, 2GB RAM		
	CPU, %	Process, MB	Thread, MB	CPU, %	Process, MB	Thread, MB	CPU, %	Process, MB	Thread, MB
ODBC	3.1	6	0.61	2.4	5.8	0.61	4.8	5.8	0.62
EJB	0.1	25.4	0.77	0.06	25.1	0.77	0.02	25.1	0.77
CORBA	0.31	25.8	0.78	0.18	25.6	0.79	0.47	25.6	0.77
DNS	0.59	3.2	0.3	0.02	3	0.3	0.17	3	0.3
WAP (secured)		4.94	0.42		4.68	0.44		4.56	0.43
WAP(not ecured)		4.91	0.22		4.78	0.035		4.57	0.15
C-Vusers	0	3	0.29	0	2.7	0.28	0	2.7	0.28
VB Script Vuser	0	5.4	0.41	0	5.1	0.4	0	5.1	0.4
MMS	95.6	10.6	3.97	109.2	16.5	4.05	177.8	12.3	3.99
MAPI	0.84	6	0.37	4.66	5	0.34	0	4.8	0.35
IMAP	0.78	3.2	0.4	0.51	3.1	0.4	0.62	3.1	0.4
CTLIB		4.15			3.8			4	
DBLIB		4.2			5.5			3.4	
INFORMIX		3.78			3.88			4.97	
LDAP	0.18	3.3	0.35	0.16	3.2	0.35	0.22	3.1	0.36
REAL (RTSP)		5.8	0.5		5.9	0.5		7.2	0.6
SMTP		3.13	0.35		2.8	0.35		3	0.35
WEB-GUI	12.8	6	1.95	12.8	5.7	1.95	12.4	5.7	1.93
VB VUSER	0	5.3	0.38	0	7.7	0.41	0	5	0.38
Siebel Web	0.76	4.9	0.68	0.71	4.3	0.66	0.84	4.6	0.72

注：以上数据均为每 50 虚拟用户条件下。

6.5.1　选择最优的测试机配置

通过 footprint 文档，可以看到同样的 vuser 脚本在不同配置测试机上的资源使用率是不一样的。

比如，同样的 Web 脚本在 Windows 2000，Windows XP，Windows 2003 下 CPU 占用率，thread/process 方式的 memory 使用率略有差别。

Protocol	Win2000, 1 CPU, 1GB RAM			WinXP, 1 CPU, 1GB RAM			Win2003, 2 CPU, 2GB RAM		
	CPU, %	Process, MB	Thread, MB	CPU, %	Process, MB	Thread, MB	CPU, %	Process, MB	Thread, MB
WEB (Http/Html)	2.5	4.9	0.82	2.2	4.6	0.82	2.1	4.6	0.82

Windows XP 下的每 50 Web 用户 CPU 占用率为 2.2%，进程方式运行占用内存为 4.6M，线程方式运行占用内存为 0.82MB。

Windows 2000 下的每 50 Web 用户 CPU 占用率为 2.5%，进程方式运行占用内存为 4.9M，线程方式运行占用内存为 0.82MB。

Windows 2003 下的每 50 Web 用户 CPU 占用率为 2.1%，进程方式运行占用内存为 4.6MB，线程方式运行占用内存为 0.82MB。

由于 Windows 2003s 的基准配置是 2 CPU，2GB 内存，而 Windows 2000 和 Windows XP 则是 1 CPU，1GB 内存.因此，可以看出 Web 虚拟用户在 Windows xp 下运行可以获得较高的性能表现。



需要注意的是，并不是所有的虚拟用户类型都适合在 Windows XP 下运行。比如从 footprint 中可以看到，MySQL 50 vuser 在 Windows 2000 下 CPU 占用率为 95.6%，进程模式运行占用内存 10.6MB，而在 Windows XP 下 cpu 占用率会达到 100%，进程模式运行占用内存 16.5MB。

6.5.2 客户端 CPU 瓶颈估算

还是以 Web 虚拟用户为例，如果选择 Windows XP 1 CPU，1GB 内存标准配置作为测试机，那么查看 footprint 文档，可以得出每 50 虚拟用户占用 CPU 为 2.2%。因此，在理想状态下，总共在本机可最大并发的虚拟用户为：

$$\text{Total Vuser}=100/2.2\times 50 = 2272\text{（个虚拟用户）}$$

结合实际因素考虑，操作系统本身，LoadRunner 等后台进程的运行也要占用 20% 左右的 CPU 资源，再加上 80%的瓶颈系数，实际情况下，在保证不出现 CPU 瓶颈

的前提下，Windows xp 1 CPU 1GB 内存标准配置的最大 CPU 瓶颈虚拟用户数为：

Total Vuser=100/2.2×80%×80%=1454 个虚拟用户

6.5.3　客户端网络瓶颈估算

网络瓶颈是一个重要但又经常被忽视的瓶颈因素。在负载测试执行时，必然会引起大量的网络流量，那么在当前的网络环境下，是否能够顺利地运载这些数据到达服务器呢？这需要在性能测试开始之前，进行一个网络瓶颈评估。

假设，当前的网络环境为 100MB 局域网，实际上 100MB 的计算单位是 bit，转化成我们最常用的 Byte，应该是 100/8=12.5M Byte。

一般，网络的瓶颈安全系数为 80%，那么实际上 loadrunner 可用的瓶颈带宽为：

Available Bandwidth = 12.5M×80% = 10M Bytes

单个虚拟用户的占用带宽可以使用网络抓包工具 wireshark 来进行分析。

例如，访问 www.cesoo.info 首页面的操作，使用 wireshark 进行抓包，如图 6-56 所示。

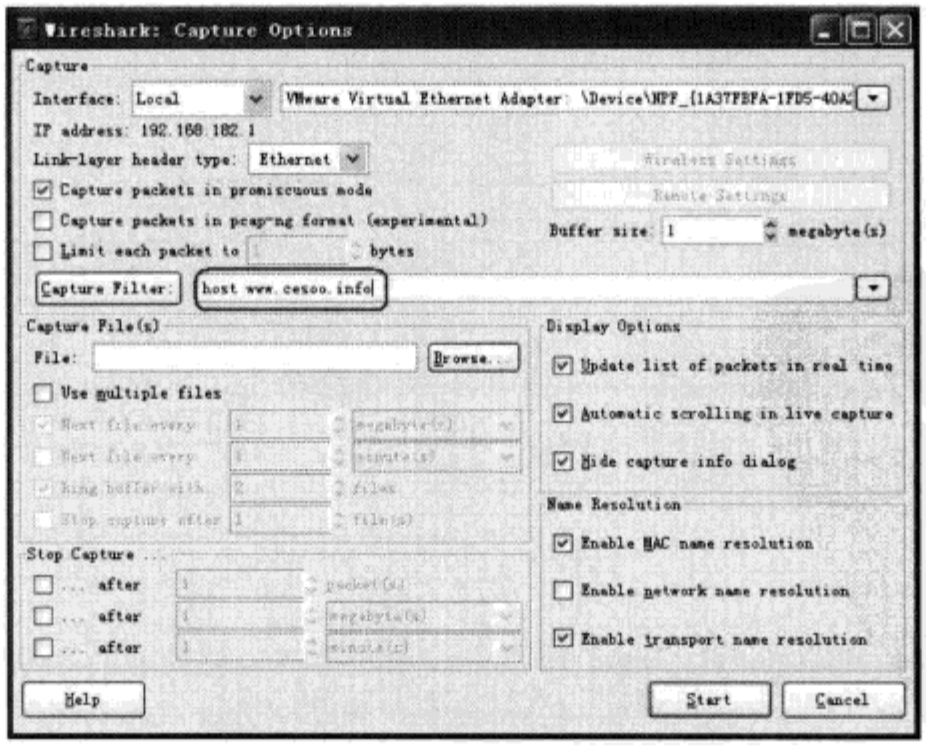


图 6-56　在录制前设置 filter

启动浏览器，访问 www.cesoo.info，会看到 wireshark 监听器已经捕捉到本机与 cesoo.info 网站之间的交互数据包，如图 6-57 所示。

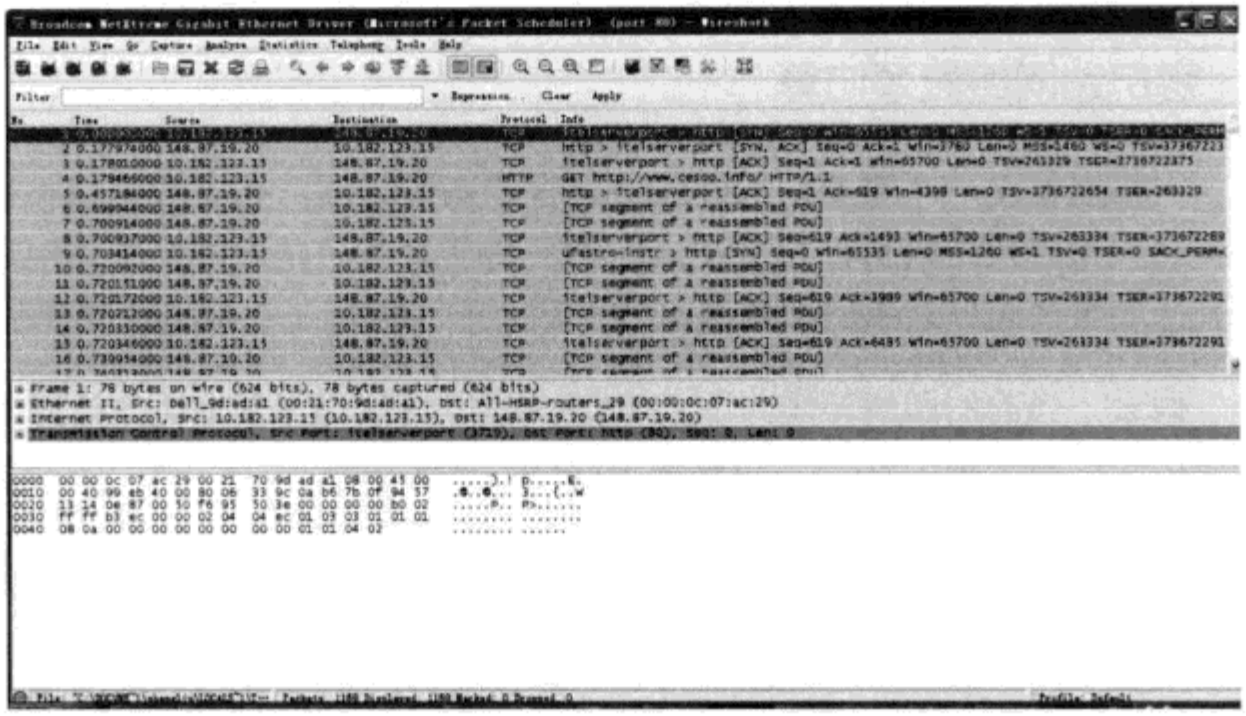


图 6-57　wireshar 捕捉到的本机与 cesoo.info 之间的网络数据包

在 wireshark 中点击“statistics”按钮，选择“summary”项，如图 6-58 所示。

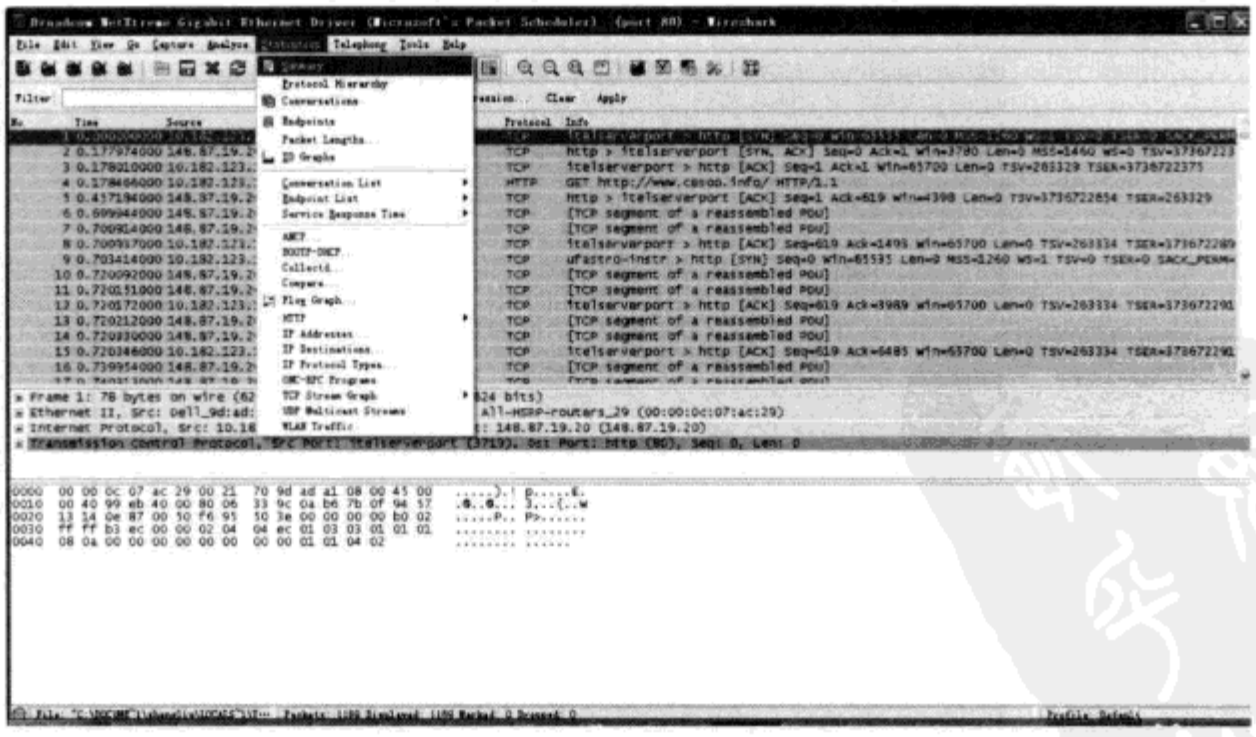


图 6-58　在 wireshark 菜单中选择 summary

可以看到有关本次访问产生的网络数据流量统计和带宽分析，如图 6-59 所示。

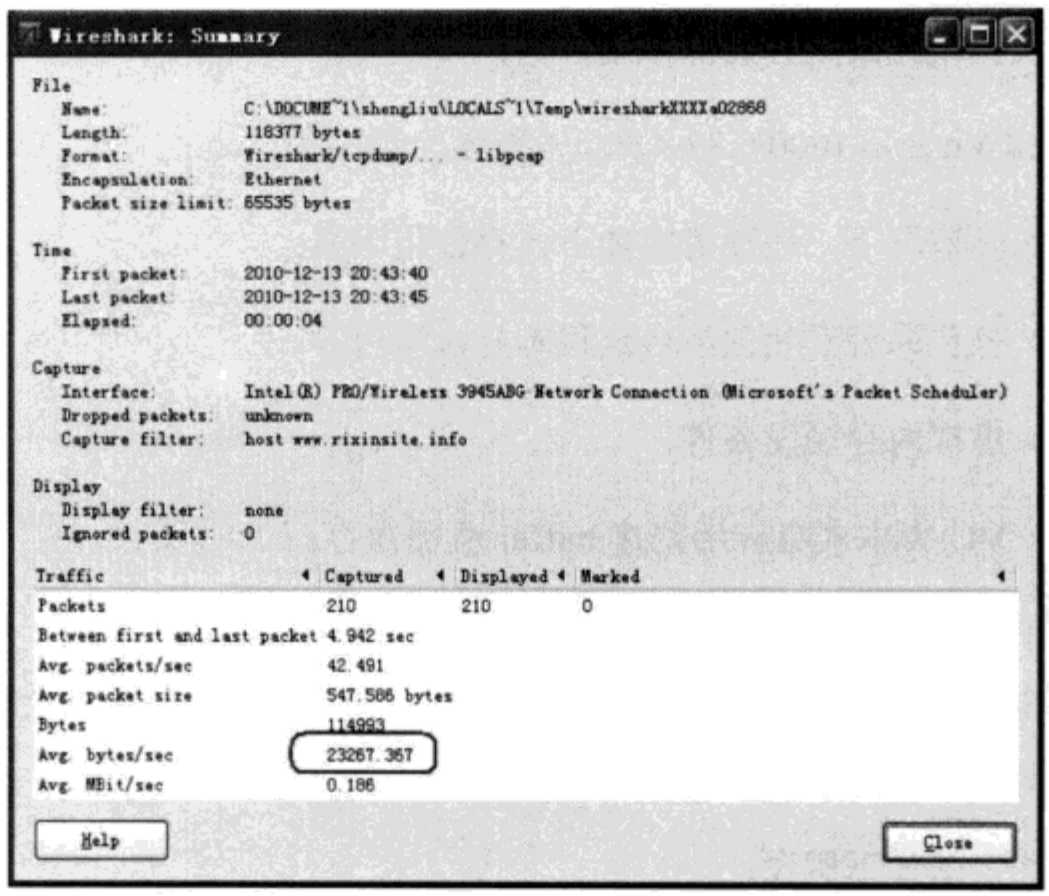


图 6-59　查看网络流量分析

可以看到本次访问有如下信息：

总共网络数据量为 114993 bytes=115KB。

占有网络带宽为 23267 bytes/sec。换算成 KB 为 24KB/S。

当前局域网内本机可并发的最大用户数为：

LoadRunner 可用带宽/单个用户带宽=10×1000/24= 416 个虚拟用户。

6.5.4　客户端内存瓶颈估算

在 Windows xp 标准配置环境下，每 50 Web 虚拟用户以进程模式运行所需内存为 4.6MB，以线程方式运行所需内存为 0.82MB。

因此，理想情况下，1GB 内存可以进程最大并发虚拟用户数为：

Total Vuser = 1000/4.6×50= 10869 个虚拟用户。

以线程模式最大并发虚拟用户数为：

Total Vuser = 1000/0.82×50 = 60975 个虚拟用户

在实际情况下，还要考虑如下内存使用因素：

（1）操作系统及相关后台进程本身占用内存。

（2）虚拟内存触发条件。

（3）VU 脚本本身应用创建 buffer 所需内存。

由于LoadRunner运行在操作系统上，操作系统内存也随着压力的发起而增大，当内存增大到一定程度的时候，虚拟内存就会被创建，这时就会进一步提高 CPU 的占用率和 IO 吞吐量，实际上形成客户端的瓶颈。

因此，做如下假设：

（1）在 LoadRunner 运行时，操作系统内存占用 25%。

（2）在物理内存使用率达到 85%时，将会创建虚拟内存（可以查看 Windows 和 Linux 相关 swap 参数设置）

（3）虚拟用户主要是处理和发送数据，一个脚本所使用的内存可以用它发送出去的网络数据大小来替代。

总共虚拟用户可用内存为：

Total Available Memory = 1000×(100-25%)×85%=638MB

每个虚拟用户占用的内存为：

Vuser Memroy = lr Vuser 执行内存+Vuser 脚本使用内存

Vuser 脚本使用内存可以通过网络流量来进行估算，如上节 wireshark 在用户

访问 www.cesoo.info 时网络抓包总流量为 115KB。那么脚本为执行此操作，至少要在内存上开辟 115KB 的数据 buffer。

因此，单个虚拟用户运行所需内存为：

Vuser Memory = 0.82+115=116KB

而在 1G 内存下，能够支持最大瓶颈并发用户数为：

Total Vuser=Total Available Memory/ Vuser Memory=638M/116KB=5500 个

6.5.5　总评估

通过以上评估计算，我们能够清楚地看到，在 Windows xp 测试机标准配置下：

（1）CPU 瓶颈最大虚拟用户并发数为 1454 个

（2）网络瓶颈最大虚拟用户并发数为 416 个

（3）内存瓶颈最大虚拟用户并发数为 5500 个

根据瓶颈取短原则，实际上，当前测试机的瓶颈虚拟用户数为 416 个。

同时，我们还可以从评估分析中得到以下结论：

（1）如果想做到 500 用户并发，应该对当前网络环境进行升级。

（2）Web 脚本不宜过长，否则数据 buffer 迟迟得不到释放，会导致本虚拟用户的内存使用过高，可能引起本机内存瓶颈。

7

第7章

Analysis——寻找系统瓶颈的得力助手

通过以上章节对 LoadRunner 的学习，我们知道了在场景运行期间查看结果数据的两种方法：

（1）Vuser 日志文件：包含对每个 Vuser 运行方案的完整跟踪。这些文件位于方案结果目录中（在以独立模式运行 Vuser 脚本时，这些文件放在 Vuser 脚本目录中）。

（2）Controller 输出：输出窗口显示有关场景运行的信息。如果方案运行失败，可以在该窗口中查找调试信息。

在本章中，我们要介绍 LoadRunner 的另外一个重要的部分——Analysis，它不仅能够直观地查看结果数据，还能生成不同格式的性能测试报告，我们可以直接将 Analysis 生成的报告提交给客户。

Analysis 的功能如下：

- “Analysis 图”有助于确定系统性能瓶颈，并可以将多个数据图合并成一个图，对多个图进行比较，从而找出数据之间的联系。
- “图数据”视图和“原始数据”视图以电子表格格式显示用于生成图的实际数据。可以将这些数据复制到外部电子表格应用程序，以进行进一步处理。
- “报告”功能可使用户查看每个图的摘要、HTML 报告或各种性能和活动报告。可以将报告创建成 Microsoft Word 文档，它会自动以图形或表格格式总结和显示测试的重要数据。

运行完场景，打开 Analysis，我们最关心的第一个问题就是性能测试通过了么？Analysis 的 Summary 视图会告诉我们答案。

7.1　Analysis 报告概要（Summary）

在 Controller 中运行完测试场景后，启动 Analysis，首先会展现 Analysis 的 Summary 页面，下面对 Summary 页面的各个部分进行介绍。

7.1.1　概要部分

Analysis 概要总结如图 7-1 所示。

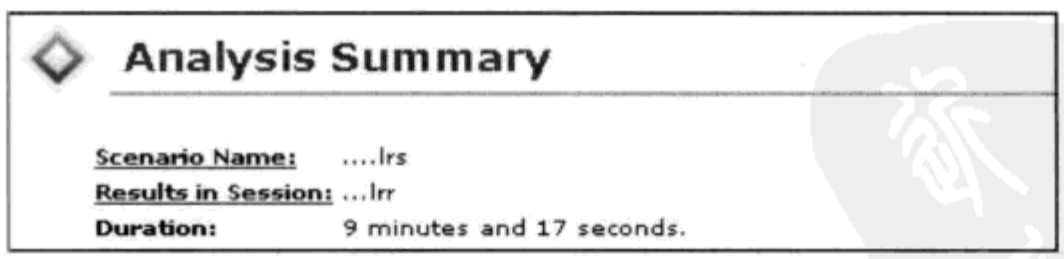


图 7-1　Analysis 概要总结

（1）Scenario Name（场景名字）

场景的名字，如果是保存过的场景，那么将会显示场景的保存路径。

（2）Results in Session（结果文件）

结果文件的名字，显示结果文件的路径。

（3）Duration（运行时间）

本场景总共运行的时间，如果脚本中有 Think Time，这里的运行时间将包含 Think Time。

7.1.2 统计部分

Analysis 统计信息如图 7-2 所示。

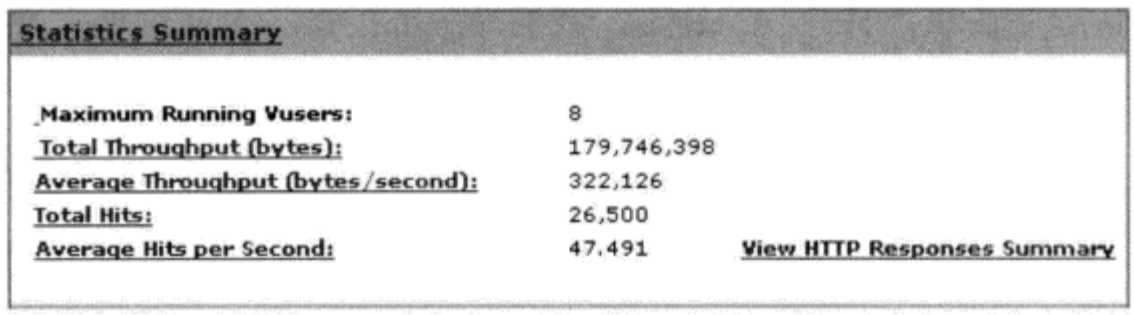


图 7-2 Analysis 统计信息

（1）Maximum Running Vusers（最大同时运行用户数）

因为 LoadRunner 有加载时间和延迟时间，这个数字通常比在场景中设置的并发用户数小。比如本例中，在 Controller 中设置 10 个虚拟用户，这里是 8 个。当然，设置了同步点之后，就不同了。

（2）Total Throughput（bytes）（网络流量）

即场景运行过程中产生的全部网络流量。单位是字节。

（3）Average Throughput（bytes/second）（网络流率）

即平均网络流率。单位是字节/秒。



网络带宽是以 bit/s 为单位的。

（4）Total Hits（总请求数）

场景运行过程中，发生的 HTTP 请求总数。

（5）Average Hits per Second（平均每秒请求数）

即为总请求数除以运行时间的值。

（6）View HTTP Responses Summary（查看 HTTP 响应的 Summary）

这是一个链接，指向一个页面标记，会在下面介绍。

7.1.3　事务统计

Analysis 事务统计信息如图 7-3 所示。

Transaction Summary

Transactions:　Total Passed: 3,500 Total Failed: 0 Total Stopped: 0　Average Response Time

Transaction Name	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
0.0 Vuser pass	5.636	6.472	11.334	0.838	7.329	500	0	0
0.1 Login URL	0.116	0.234	3.302	0.224	0.344	500	0	0
1.1 Login submit	0.786	1.117	4.641	0.299	1.382	500	0	0
Delete INBOX	0.524	0.78	4.343	0.249	1.018	500	0	0
Logout	0.078	0.183	3.278	0.2	0.303	500	0	0
Undel msgs	0.462	0.721	3.46	0.222	0.968	500	0	0
View Trash	0.432	0.642	3.665	0.234	0.897	500	0	0

图 7-3　Analysis 事务统计信息

（1）Transactions（事务）

所有 Transaction 的响应时间都列在这里。下面对各个响应时间指标一一介绍。

（2）Minimum

执行此 Transaction 的最小时间，也就是最快的。

（3）Average

平均时间，是所有 Transaction 消耗时间的算术平均值。

（4）Maximum

执行此 Transaction 的最大时间，也就是最慢的。

（5）Std.Deviation

Std 是单词 standard 的缩写，Std.Deviation 即标准方差，方差是描述一组数据偏离其平均值的情况。方差的公式如下：

$$S^2 = \frac{1}{n}\left[(x_1 - \overline{x})^2 + (x_2 - \overline{x})^2 + \cdots + (x_x - \overline{x})^2\right]$$

从数学意义上看，方差值越大，这组数据就越离散，波动性也越强；方差越小，这组数据就越聚合，波动性也就越小。



显然，方差这个数字能说明一些问题，比如我们如果试图比较同一场景的先后两次运行结果，在 Average 相差比较大的情况下，我们可以通过 Std.Deviation 值来判断这个变化是由一些随机因素引起的，还是确实调优或修改代码生效了。如果方差小，说明这些数据凝聚度高，随机因素干扰小；方差大，则相反。因此可以说在事务统计信息里，相关方差值越小，这组数据越好，越有说服力。

（6）90 Percent

在 Controller 运行场景时，并不会显示这个值，因为它是对一系列数据进行计算的结果。是 90% Transaction 消耗的时间，比如上面执行了 500 次 Transaction，90%就是 450 个 Transaction 的平均时间，通常这个指标比单纯的平均值更能说明系统问题。

（7）Pass（通过）

通过的 Transaction 个数。通过与否的标准由脚本的逻辑控制。

（8）Fail（失败）

失败的 Transaction 个数。

（9）Stop（停止）

如果在执行场景时，用户手工停止了场景的执行，Transaction 没有自己的状态，那么就是 Stop。

7.1.4　HTTP 响应统计

Analysis 中 HTTP 响应状态码统计如图 7-4 所示。

HTTP Responses Summary

HTTP Responses	Total	Per second
HTTP_200	26,500	47.491

图 7-4　Analysis 中 HTTP 响应状态码统计

此视图只有 Web Vuser 才有，它反映了 Web Server 的处理情况。

HTTP 返回码为 200，是正常状态。我们还记得在统计图中的 Hits（总请求数）为 26500 个，在此图中，200 返回码的 Response 有 26500 个，说明 HTTP 的请求都成功了。Web Server 是没有问题的。

如果测试结果没有达到我们既定的性能指标，我们会有第二个问题：那么问题出现在哪里？Analysis 提供的多个图会帮助我们去寻找答案。

7.2　Analysis 标准图

7.2.1　Vuser 图

点击展开 Graph 树中的 Vuser 图，如图 7-5 所示。

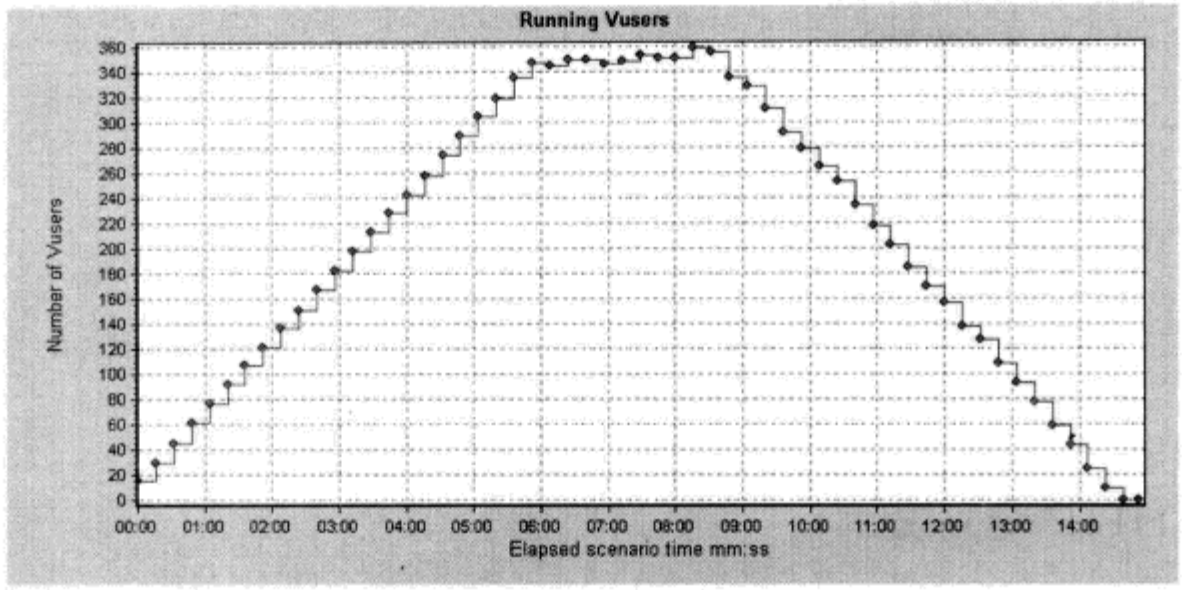


图 7-5 Analysis 虚拟用户图

Vuser 图展开之后，我们可以看到场景刚开始的时候，Vuser 数目逐渐增加，大概在接近 6 分钟的时候，一共有 360 个并发用户同时工作。

需要注意的是，在 Vuser 图中包含所有的 Vuser，包括失败的 Vuser。可以设置过滤器（filter）来查看我们关注的那一部分 Vuser，可以是分组观察，也可以单个观察。

设置过滤器的方法是：右键单击图，在弹出的菜单中选择“Set filter/Group by”，在打开的对话框中选择过滤条件，如图 7-6 所示。

在“Filter condition”中选择“Vuser End Status”，Criteria 选择“=”，Values 选择“Passed”，这就过滤了那些失败的用户了。我们也可以选择观察某个时间段的 Vuser，比如第 6 分钟以后的 Vuser 情况，就在“Scenario Elapsed Time”中选择一个时间段，如图 7-7 所示。

如果我们想回过头去看初始图，可以清除过滤器（filter），选择右键菜单中的“Clear filter/Groups”即可。

7.2.2 事务图

首先，看“Transaction Performance Summary”图，确认哪个事务的响应时间

比较大，超出了我们的标准。

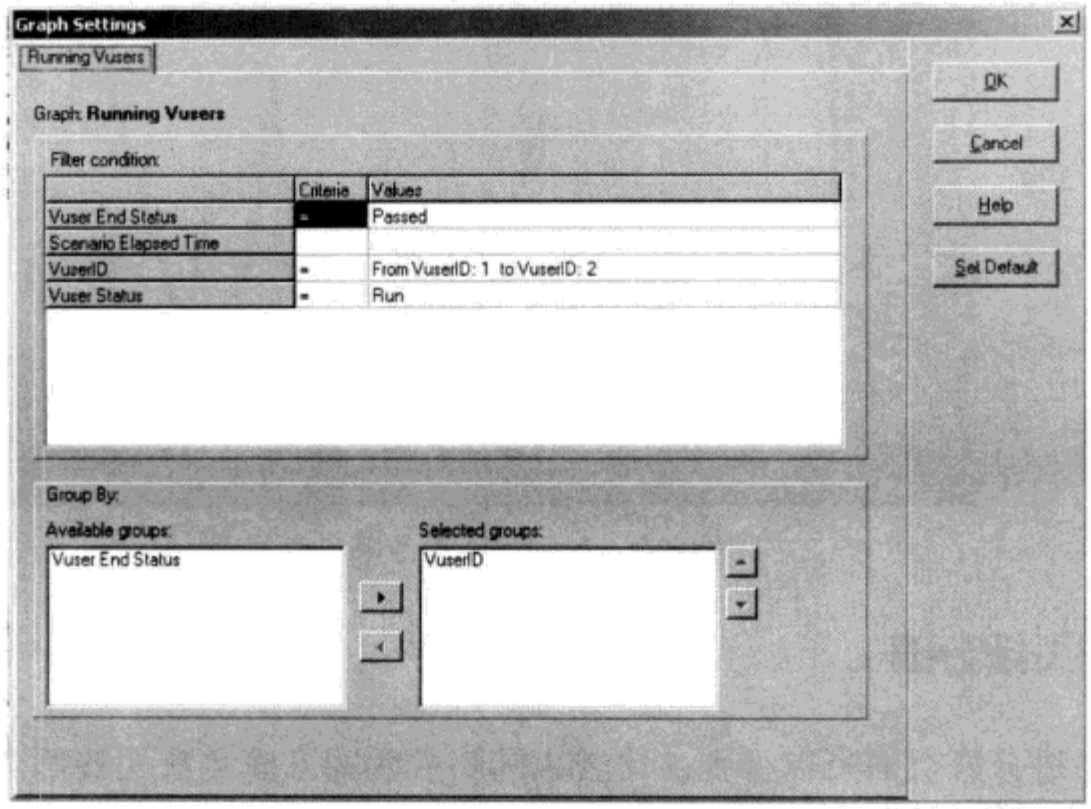


图 7-6　Analysis 设置过滤条件

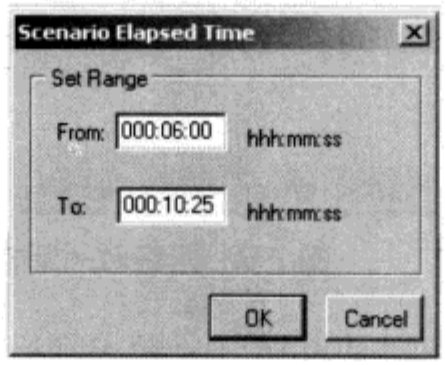


图 7-7　Analysis 设置截取时间

然后，我们再看“Average Transaction Response Time”，观察 Login 在整个场景运行中每一秒的情况。从图 7-8 中可以看出，Login 事务的响应时间并不是一直都比较高，只是随着用户数的增加，响应时间才明显增加的。

为了定位问题，明白为什么 Login 事务的响应时间比较长，现在我们要分解 Login 事务，分析该页面上每一个元素的性能。在图 7-8 中，选择要分解的事务曲线，然后单击鼠标右键，选择“Web Page Breakdown for login”。

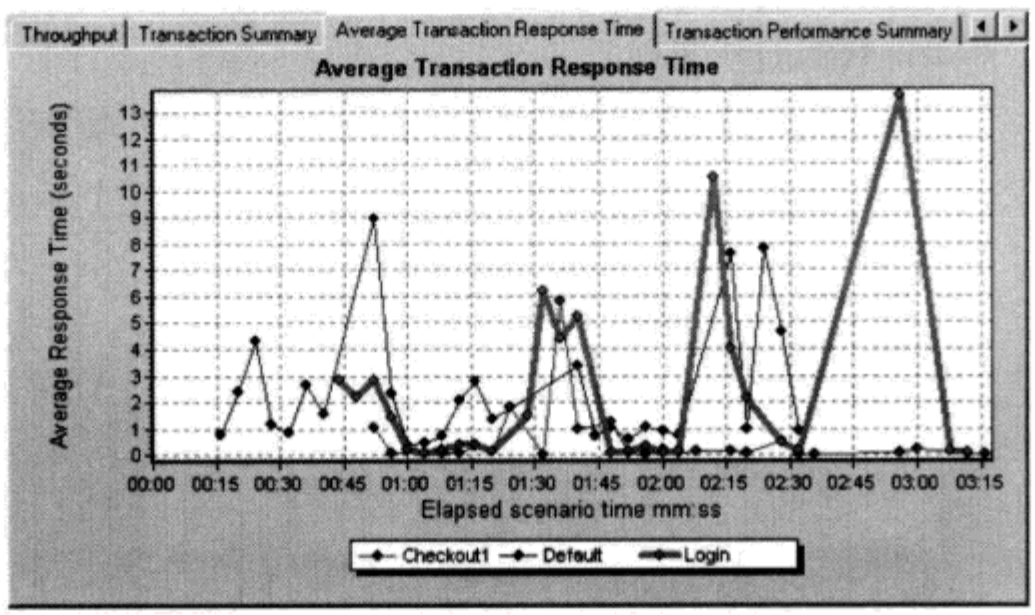


图 7-8 Analysis 事务响应时间图

7.2.3 网页细分图

通过分解页面可以得到：比较大的响应时间到底是页面的哪个组件引起的？问题出在服务器上还是网络传输上？

这里要解释一下各个时间的含义（比如：DNS 解析时间、连接时间、接收时间等）。



图 7-9 网络时间解析图

下面简单说一下浏览器从发送一个请求到最后显示的全过程。

① 浏览器向 Web Server 发送请求，一般情况下，该请求首先发送到 DNS Server，把 DNS 名字解析成 IP 地址。解析过程的时间就是 DNS 解析时间（DNS resolution）。这个度量时间可以确定 DNS 服务器或者 DNS 服务器的配置是否有问题。如果 DNS Server 运行情况比较好，该值会比较小。

② 解析出 Web Server 的 IP 地址后，请求被送到了 Web Server，然后浏览器和 Web Server 之间需要建立一个初始化连接。建立该连接的过程就是连接时间（connection）。这个度量时间可以简单地判断网络情况，也可以判断 Web Server

是否能够响应这个请求。如果正常，该值会比较小。

③ 建立连接后，从 Web Server 发出第一个数据包，经过网络传输到客户端，浏览器成功接收到第一字节的时间就是 First Buffer。这个度量时间不仅可以表示 Web Server 的延迟时间，还可以表示出网络的反应时间。

④ 从浏览器接收到第一个字节起，直到成功接收到最后一个字节，下载完成为止。这个度量时间可以判断网络的质量（可以用 size/time 比来计算接收速率），其他的时间还有 SSL Handshaking（SSL 握手协议，用到该协议的页面比较少）、ClientTime（请求在客户端浏览器延迟的时间，可能是由于客户端浏览器的 Think Time 或者客户端其他方面引起的延迟）、Error Time（从发送了一个 HTTP 请求，到 Web Server 发送回一个 HTTP 错误信息所需要的时间）。

熟悉了以上各个时间的含义后，我们开始看分解页面的图形。

首先看“Downlaod Time Breakdown”，可以看出 Login 事务分解的各个组件的大小，以及各个组件的下载时间。

从图 7-10 可以看出，Login 页面由 5 个组件组成，其中 next.gif 下载用的时间最长，这说明有可能是 next.gif 图片尺寸过大引起的，我们还需要其他数据来进一步判断。

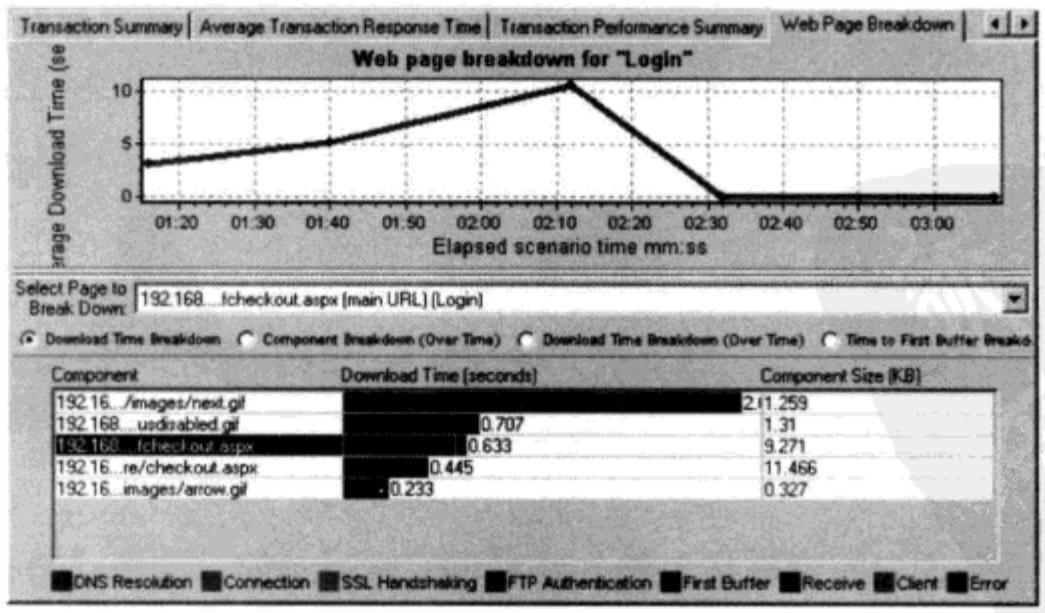


图 7-10　Analysis 网页细分图

选择“Component Breakdown（Over Time）”，可以以图形曲线的形式看出各个组件在场景运行中每秒钟的下载时间（如图 7-11 所示），比较具体。

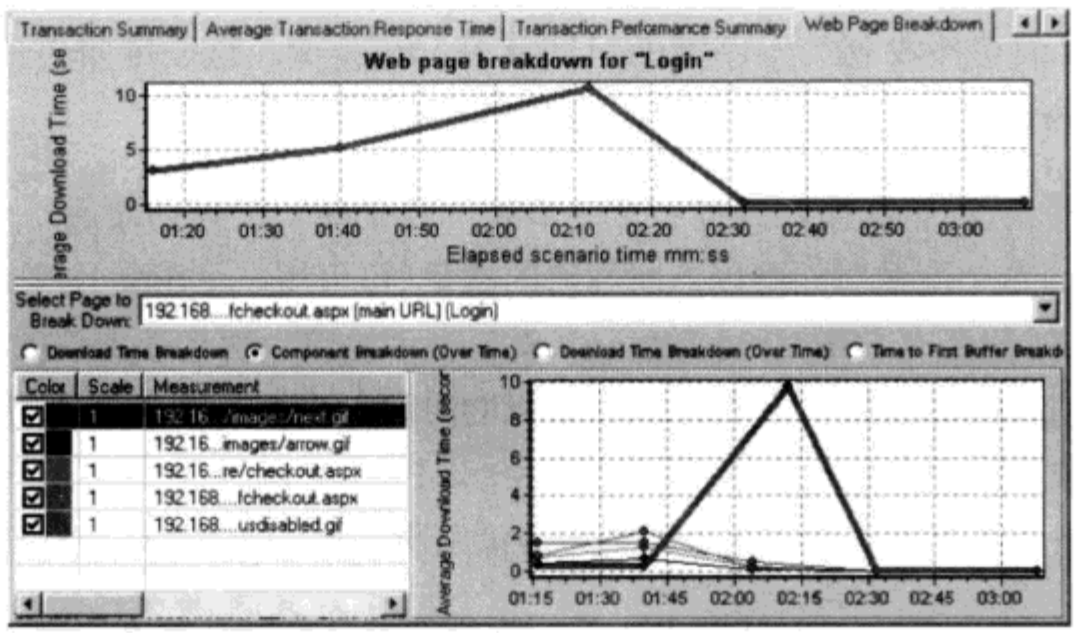


图 7-11　Login 事务组件细分图

然后再选择“Download Time Breakdown（Over Time）”，从中可以看出在场景运行中的每一秒钟，组件用在传输上的各部分的时间（如图 7-12 所示）。

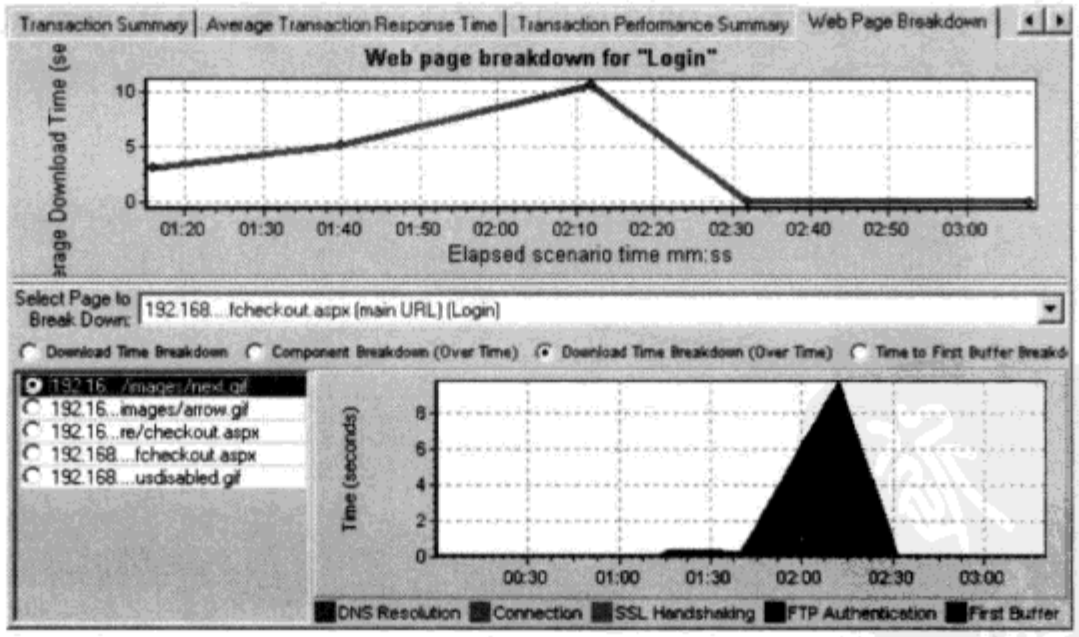


图 7-12　Login 事务下载时间细分图

为了确认问题到底是由服务器还是网络引起的，选择“Time to First Buffer Breakdown”（如图 7-13 所示），可以看出 Server 时间比 Network 时间要高得

多，从而确定问题是服务器引起的。然后我们就可以参考 Web Server 的相关图表，来确定问题是由服务器的哪个部分引起的。遵从这样的步骤，可以一步步地接近问题源；如果问题由网络引起，可以参考 Network 相关的图表，确定什么样的网络问题是性能的瓶颈。同时可以参考“Time to First Buffer Breakdown(Over Time)”。

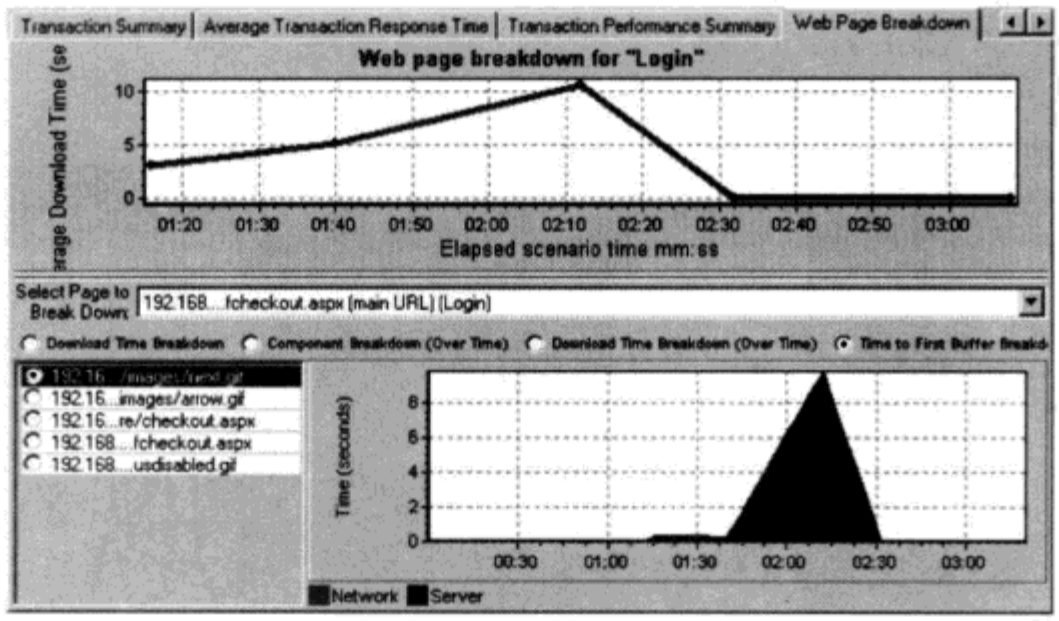


图 7-13　First Buffer 细分图

7.3　交叉结果和合并图

使用 Analysis 可以将同一场景的两个图中的结果合并到一个图中。通过合并，可以一次比较几个不同的度量，以发现它们之间的联系。例如，你可以制作一个合并图，以已用时间的函数的形式显示网络延迟和正在运行的 Vuser 的数量。

要合并图，这些图的 *X* 轴的度量单位必须相同。例如，我们可以合并“Web 吞吐量和每秒点击次数”图，因为它们具有公用的 *X* 轴：场景的已用时间。该下拉列表仅显示 *X* 轴与当前图相同的活动图。

我们可以对 Windows Resources 图进行这样的操作。首先打开 Windows Resources 图表，然后在图表上单击鼠标右键，选择“Merge Graphs”，如图 7-14

所示。

出现“Merge Graphs”对话框，如图 7-15 所示。

在合并图时有 3 种方式可选：

- 叠加（Overlay）；
- 平铺（Tile）；
- 关联（Correlate）。

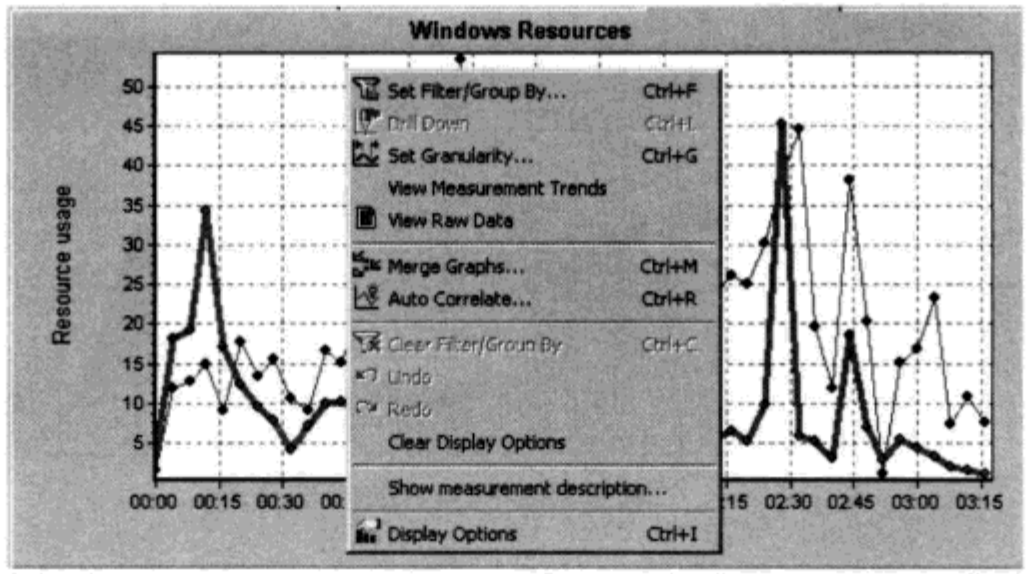


图 7-14　在右键菜单中选择合并图

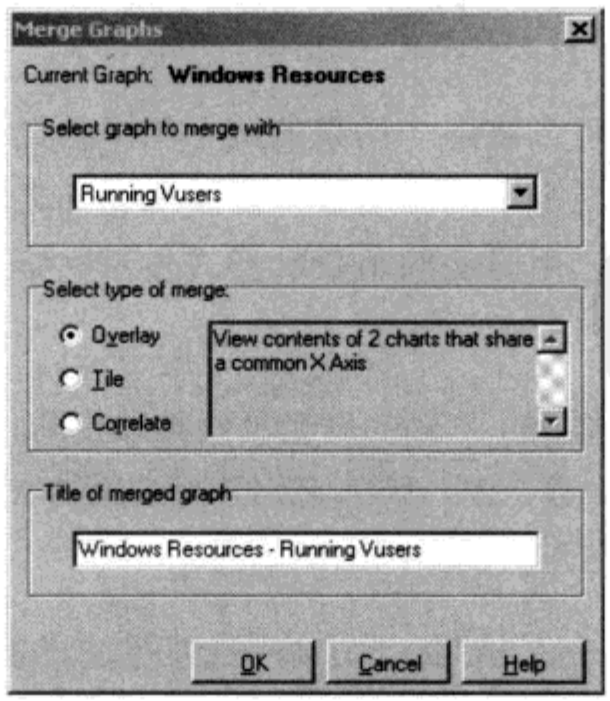


图 7-15　合并详细信息对话框

7.3.1　叠加

重叠共用同一个 *X* 轴的两个图的内容。合并图左侧的 *Y* 轴显示当前图的值，右侧的 *Y* 轴显示已合并图的值。叠加图的数量没有限制。叠加两个图时，这两个图的 *Y* 轴分别显示在图的右侧和左侧。覆盖两个以上的图时，Analysis 只显示一个 *Y* 轴，相应地缩放不同的度量。

选择第一项“Overlay”，出现如图 7-16 所示的图，这样是把两个图进行了合并，两条曲线的纵轴共用一个原点，横轴还是时间轴。

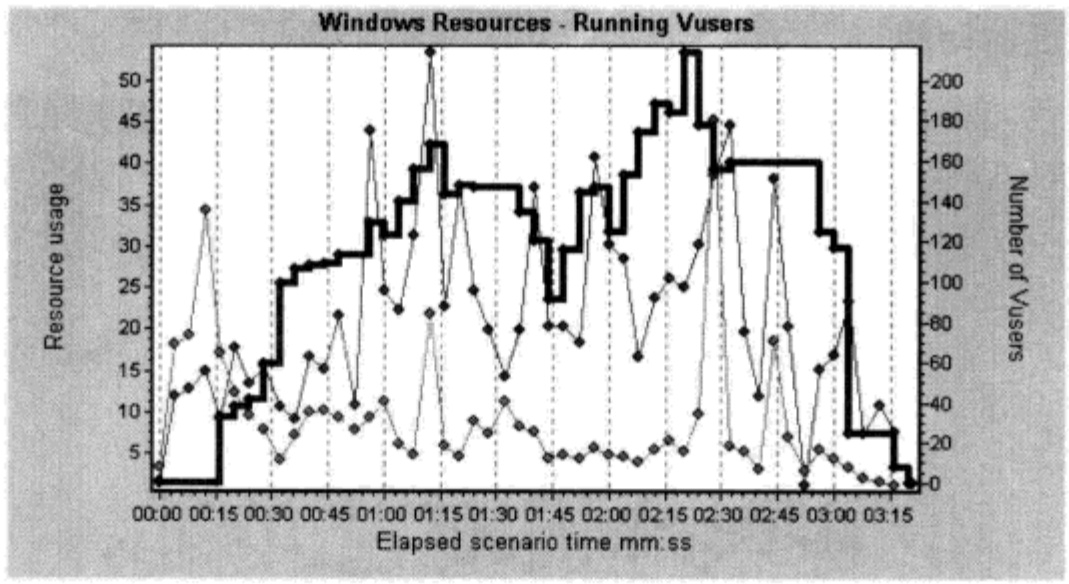


图 7-16　叠加图

在叠加图中，我们可以看到在 2:22 分时虚拟用户数达到最高 200，而在 2:32 秒左右，Windows 的资源使用率才达到峰值，这说明负载压力从客户端传递到服务器并体现出来，是有时间延迟的。

? 思考

对一个基于 Java 的 Web 应用系统进行负载测试，在生成的虚拟用户数和服务器内存占用率的关联图中，如果虚拟用户数曲线从峰值下滑的同时，而内存占用率却一直保持在峰值不变，这是否能得出被测服务器上的 Java 程序存在内存泄露的问题？（Java 内存泄露的问题可参看 Java 虚拟机原理相关文章。）

如果并发虚拟用户数已经回落到零，而内存占用率曲线还在继续上扬，这是否能得出 Java 内存泄露结论？

7.3.2 平铺

查看在平铺布局（一个位于另一个之上）中共用同一个 *X* 轴的两个图的内容。

选择第二项“Tile”，出现如图 7-17 所示的图，这样是把两个图进行了合并，两条曲线的纵轴不再共用一个原点，Vusers 的原点在 Windows Resouces 的上面，横轴还是时间轴。

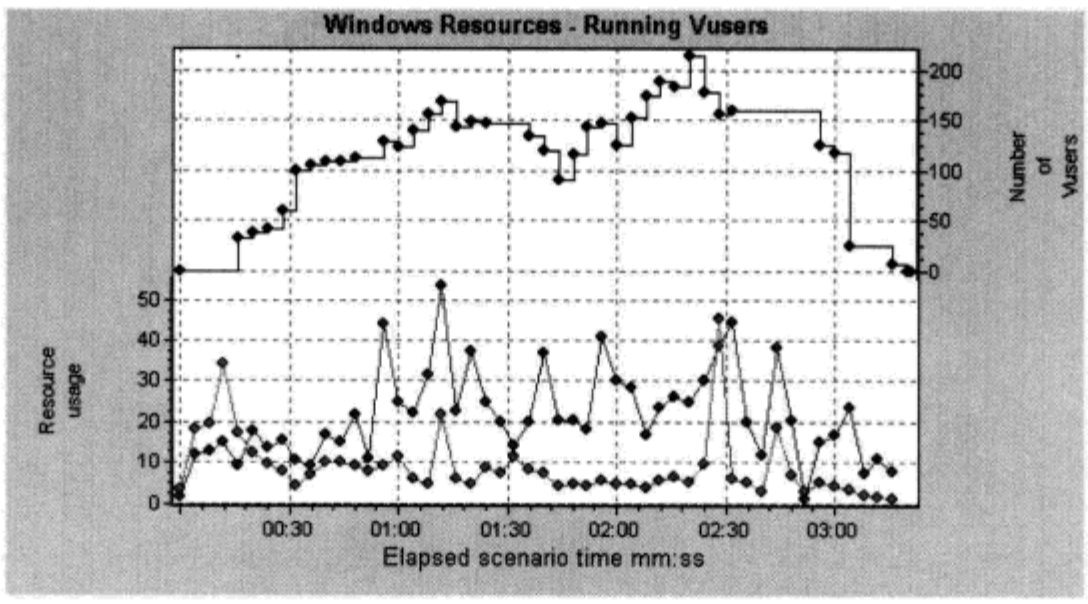


图 7-17 平铺图

7.3.3 关联

绘图时区分两个图彼此的 *Y* 轴。活动图的 *Y* 轴变为合并图的 *X* 轴，被合并图的 *Y* 轴作为合并图的 *Y* 轴。

Windows Resources 图和 Vusers 运行图彼此关联如图 7-18 所示。

从上面的虚拟用户和 Windows Resources 关联图中我们可以看出，虚拟用数的增长和 Windows CPU 使用率（绿线）的增长，存在着一个大致的线性关系。

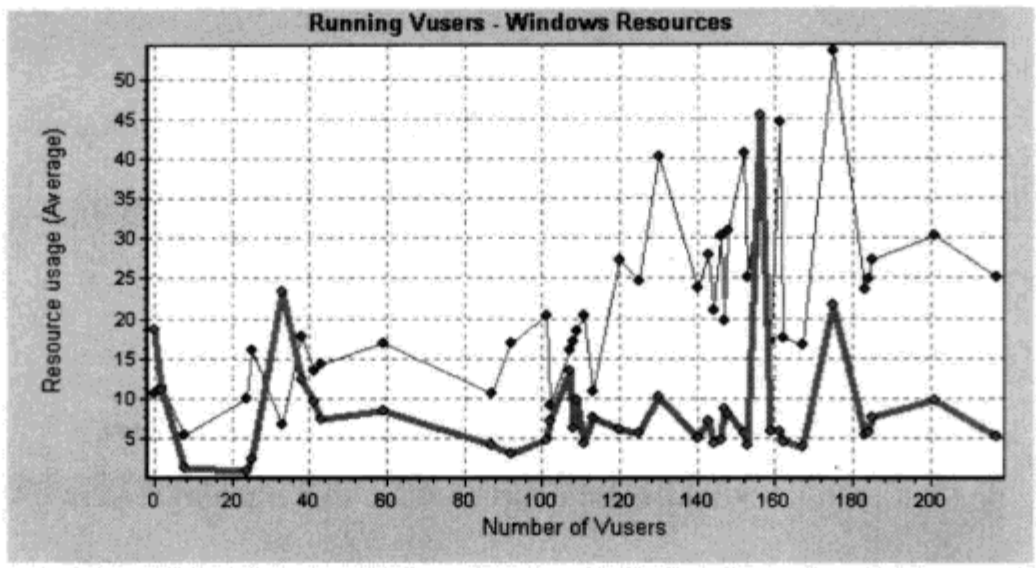


图 7-18　关联图

? 思考

线性关系中的起伏波浪可能由于什么因素引起的？为什么在虚拟用户数增加的同时，资源的使用率有时还会下降？

7.4　生成测试报告

运行场景后，我们可以查看对系统性能进行汇总的报告。Analysis 提供 3 种格式的报告：HTML 格式、Word 格式和水晶报表。

7.4.1　创建 HTML 格式报告

使用 Analysis 可以为方案的运行创建 HTML 报告，它将为每个打开的图分别创建一个报告和一个摘要报告。该摘要报告与你从 Analysis 窗口中访问的摘要报告相同。该报告还提供指向特定 Excel 文件（包含图数据）的链接，如图 7-19 所示。

要创建 HTML 报告，请执行下列操作：

① 打开要包含在报告中的所有图。

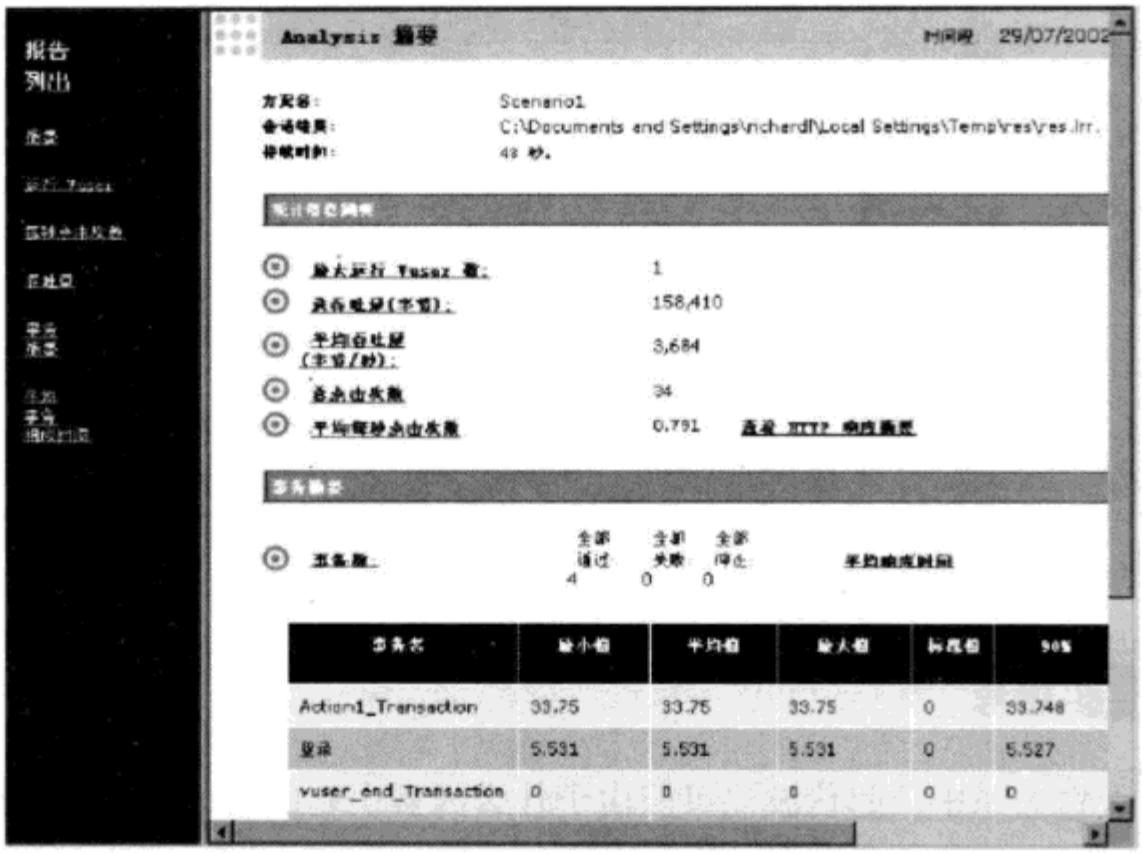


图 7-19 Analysis 创建 HTML 格式报告

② 选择“报告”>“HTML 报告”，或者单击“创建 HTML 报告”，将打开“选择报告文件名和路径”对话框。

7.4.2 创建 Word 格式报告

① 打开要包含在报告中的所有图。

② 选择“报告”>“Microsoft Word Report”。

创建水晶报表：LoadRunner 的水晶报表是一个更为细致的报告，分为以下类别：

- 活动报告；
- 性能报告。

活动报告提供关于虚拟用户和场景运行的信息，包括开始时间、结束时间等详细信息。活动报告又有“Scenario Execution”（场景执行）、“失败的交易”

（Failed Transaction）和“失败的虚拟用户”（Failed Vuser）三种。性能报告可分析每个 Transaction（事务）所用的时间，其中有“数据点”、“详细事务”和“事务性能”（按 Vuser）。

7.4.3　高级——数据点报告

使用 LoadRunner，我们可以记录自己的分析数据。我们可以指示 LoadRunner 记录外部函数或变量（也称为方案运行期间的“数据点”）的值。LoadRunner 将使用收集的数据创建数据点图和报告。

设置数据点时，会将 **lr_user_data_point** 函数（GUI Vuser 的 **user_data_point**）包括在 Vuser 报告中。具体的使用方法我们在 Controller 一章中已经介绍过了。

“数据点”图显示方案运行期间数据点的值。*X* 轴表示从开始运行以来已用的秒数；*Y* 轴显示记录的每个数据点语句的值。

“数据点”报告是一种性能报告，它列出数据点的名称、值以及记录该值的时间。值将对每个组和 Vuser 都显示。

事务性能摘要（按 Vuser）报告

“事务性能摘要（按 Vuser）”报告是一种性能报告，显示每个 Vuser 在方案运行期间执行事务所需的时间。该报告表明事务是否成功，以及每个 Vuser 的最小、最大和平均时间。当方案具有多种不同类型的 Vuser，并且需要具体描述每种类型的性能时，此报告十分有用。



第8章

欲善其事先利其器——VU脚本开发实战

VU 开发脚本是我们使用 LoadRunner 做性能测试中一个重要的步骤，这里涉及软件系统架构、计算机编程技巧和 VU 本身的功能特性。在本章，我们将结合邮件系统，介绍如何使用 VU 生成基于 SMTP/IMAP 协议的脚本。

我们将按照如下的思路完成脚本。

① 熟悉 SMTP/POP3/IMAP 协议规范和原语。

② 使用 VU 录制邮件收发操作，分析 VU 脚本对标准 SMTP/IMAP 协议的封装与实现机理。

③ 使用 VU Java 模板用户嵌入 Java message 接口，实现邮件收发，并以此介绍 VU Java 模板用户的使用方法，以及 Java classpath 等相关设置。

8.1　邮件服务 SMTP/IMAP 协议介绍

电子邮件是我们日常工作中经常使用到的一种交流方式，它是 Internet 应用最广的一种服务。

8.1.1　电子邮件的工作原理

电子邮件是 Internet 上最为流行的应用之一。如同邮递员分发投递传统邮件一样，电子邮件也是异步的，也就是说，人们是在方便的时候发送和阅读邮件的，无须预先与别人协同。与传统邮件不同的是，电子邮件既迅速，又易于分发，而且成本低廉。另外，现代的电子邮件消息可以包含超链接、HTML 格式文本、图像、声音甚至视频数据。

电子邮件的工作过程遵循客户-服务器模式。每份电子邮件的发送都要涉及发送方与接收方，发送方构成客户端，而接收方构成服务器，服务器含有众多用户的电子信箱。发送方通过邮件客户程序，将编辑好的电子邮件向邮件服务器（SMTP 服务器）发送。邮件服务器识别接收者的地址，并向管理该地址的邮件服务器（IMAP 或 POP3 服务器）发送消息。邮件服务器将消息存放在接收者的电子信箱内，并告知接收者有新邮件到来。接收者通过邮件客户程序连接到服务器后，就会看到服务器的通知，进而打开自己的电子信箱来查收邮件。

8.1.2　SMTP 协议介绍

简单邮件传送协议（SMTP）是 Internet 电子邮件系统首要的应用层协议。它使用由 TCP 提供的可靠的数据传输服务把邮件消息从发信人的邮件服务器传送到收信人的邮件服务器。SMTP 协议服务的默认端口是 25。

SMTP 协议与人们用于面对面交互的礼仪之间有许多相似之处。首先，运行在发送端邮件服务器主机上的 SMTP 客户，发起建立一个到运行在接收端邮件服务器主机上的 SMTP 服务器端口号 25 之间的 TCP 连接。如果接收邮件服务器当

前不在工作，SMTP 客户就等待一段时间后再尝试建立该连接。这个连接建立之后，SMTP 客户和服务器先执行一些应用层握手操作。就像人们在转手东西之前往往先自我介绍那样，SMTP 客户和服务器也在传送信息之前先自我介绍一下。在这个 SMTP 握手阶段，SMTP 客户向服务器分别指出发信人和收信人的电子邮件地址。彼此自我介绍完毕之后，客户发出邮件消息。SMTP 可以指望由 TCP 提供的可靠数据传输服务把该消息无错地传送到服务器。如果客户还有其他邮件消息需发送到同一个服务器，它就在同一个 TCP 连接上重复上述过程；否则，它就指示 TCP 关闭该连接。

让我们看一个客户（C）和服务器（S）交互的例子。前面标以“C:”的文本行是名为 Mike 的客户端发送的请求，Mike 想给 Rose 发送一封“I love you”的电子邮件情书，前面标以“S:”的是 cesoo.com 服务器的回应。以下传输内容在 TCP 连接建立之后马上发生。

```
S:220 cesoo.com
C:HELO
S:250 Hello Mike,pleased to meet you
C:MAIL FROM: Mike@cesoo.com
S:250 Mike@cesoo.com ... Sender OK
C:RCPT TO: rose@cesoo.com
S:250 rose@cesoo.com...Recipient OK
C:DATA
S:354 Enter mail,end with "." on a line by its self
C: I love you, Rose
C: .
S:250 Message accepted for delivery
C:QUIT
S:221 cesoo.com closing connection
```

觉得难以置信吗？这不是在写言情小说，而确确实实是客户端和邮件服务器通过 SMTP 协议在网络上交互的内容，网络协议就这么简单。客户总共发出了 5 个命令，分别为：HELO、MAIL FROM、RCPT TO、DATA 和 QUIT。这些命令又叫做**原语**，可理解为应用协议层上最原始最小的命令颗粒。服务器给每个客户端的命令发回应答，其中每个响应都由返回码和一些英语解释构成。这里需要指

出的是，SMTP 使用持久连接，也就是说，如果客户端有多个邮件消息需发送到同一个邮件服务器，那么所有这些消息可以在同一个 TCP 连接中发送。对于其中的每一个消息，客户端以一个新的“HELO”命令开始整个消息发送过程，但是 QUIT 命令要等到所有消息都发送完之后才发出。

8.1.3　POP3 协议介绍

大家一听这个 POP，读起来有点像中文中的泡泡，其实这是一个英文术语的缩写。POP 的全称是 Post Office Protocol，即邮局协议，用于电子邮件的接收，它使用 TCP 的 110 端口。现在常用的是第三版，所以简称为 POP3。POP3 仍采用 Client/Server 工作模式，Client 被称为客户端，一般我们日常使用电脑都是作为客户端，而 Server（服务器）则是 POP3 的邮件服务器。举个形象的例子：Server（服务器）是许多小信箱的集合，就像我们所居住楼房的信箱结构，而客户端就好比是一个人拿着钥匙去信箱开锁取信，一样的道理。

POP3 和上面的 SMTP 协议一样，其实现也是一个客户端与服务器的对话过程。

当我们单击了电子邮件软件中的收取按钮后，电子邮件软件首先会调用 DNS 协议对 POP 服务器进行解析 IP 地址，当 IP 地址被解析出来后，邮件程序便开始使用 TCP 协议连接邮件服务器的 110 端口，因为 POP 服务器是比较忙的，所以在这个过程中我们相对要等比较长的时间。当邮件程序成功地连上 POP 服务器后，其先会使用 USER 命令将邮箱的账号传给 POP 服务器，然后再使用 PASS 命令将邮箱的账号传给服务器。当完成这一认证过程后，邮件程序使用 STAT 命令请求服务器返回邮箱的统计资料，比如邮件总数和邮件大小等，然后 LIST 命令便会列出服务器里邮件数量。接下来邮件程序就会使用 RETR 命令接收邮件，接收一封后便使用 DELE 命令将邮件服务器中的邮件置为删除状态。当使用 QUIT 命令时，邮件服务器便会将置为删除标志的邮件给删了。通俗地讲，邮件程序从服务器接收邮件，其实就是一个对话过程，POP 协议就是用在电子邮件中的一门语言。

8.1.4 IMAP 协议介绍

用户使用 POP3 把邮件消息下载到本地机之后，就可以把它们移动到本地创建的文件夹中。用户然后可以删除邮件，移动邮件，按发信人名字或消息主题搜索邮件等。然而，所有这些邮件操作都是在本机上完成的。这对于游动的用户却构成了问题，游动用户更愿意在远程邮件服务器主机上维护邮件夹，这样从任何主机都可以访问它，使用 POP3 是不可能做到这一点的。

这时 IMAP 协议就应运而生了，同样也是邮件接收协议，但是 IMAP 却比 POP3 复杂得多，因为 IMAP 提供的特性比 POP3 多出不少。IMAP 被设计成允许用户像对待本地邮箱那样操纵远程邮箱。具体地说，IMAP 使得收信人能够在自己的邮件服务器主机中创建并维护多个存放邮件的文件夹。他们可以把邮件存入文件夹，也可以将邮件从一个文件夹转移到另一个文件夹，还可以在这些远程邮件夹中搜索匹配特定准则的邮件消息。IMAP 的实现比 POP3 的实现复杂得多，原因之一就是 IMAP 服务器必须为每个用户维护一个文件夹层次结构。某个用户使用不同的客户端相继访问自己的 IMAP 服务器时，这个 IMAP 服务器为该用户维护并同步相应的状态。POP3 服务器则相反，一旦用户退出当前的 POP3 会话，它们就不再为用户维护状态信息了。

8.2 VU 的 SMTP Vuser 对 SMTP 协议的封装及实现

上面我们分析了 SMTP/POP3/IMAP 各自的协议原理及相应实现原语，下面我们通过对比 SMTP 协议原语与 VU 录制发送邮件操作而产生的脚本函数，来看一下 VU 是如何对 SMTP 协议进行封装的。在录制之前，需保证本机已经安装邮件客户端，本例中采用 Outlook 2003，并且已经配置好指向相应的邮件服务器。

配置步骤如下：

① 在控制面板中，双击“邮件”图标，弹出如图 8-1 所示的对话框。

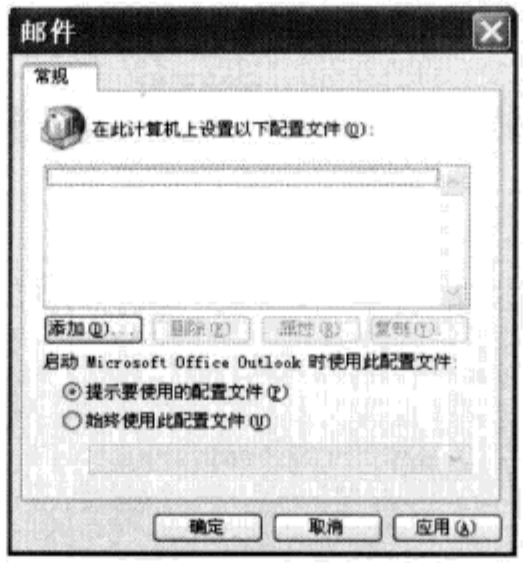


图 8-1　邮件配置文件列表

② 单击“添加”按钮，输入一个配置文件名，比如“cesooMail”，然后单击“确定”按钮，弹出如图 8-2 所示的对话框。

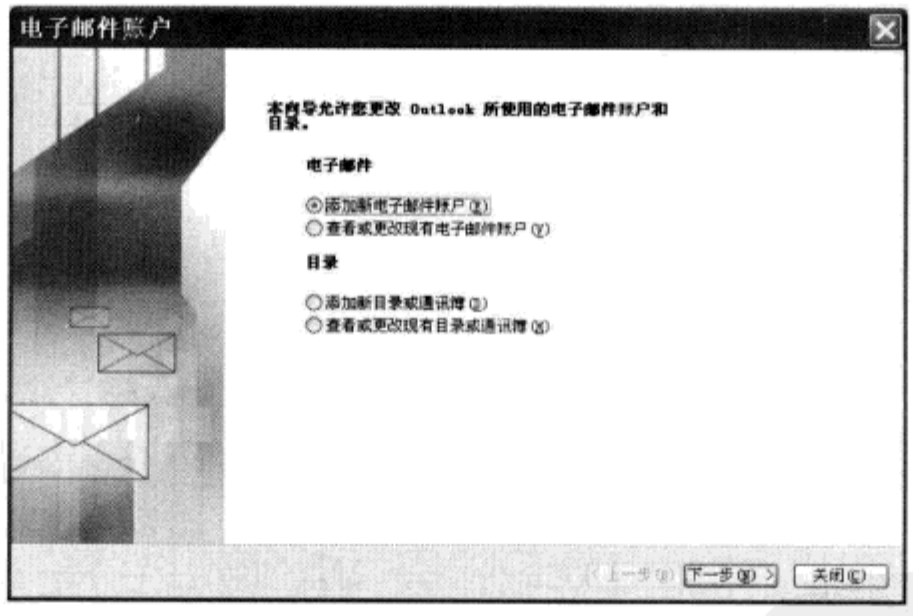


图 8-2　邮件配置向导第一步

③ 保持图中的默认选项不变，单击“下一步”按钮，进入邮件配置向导第二步，如图 8-3 所示。

④ 勾选“IMAP”类型，单击“下一步”按钮，进入邮件账户配置信息对话框，在对话框里输入相应的账号和 SMTP/IMAP 服务器信息，注意这里的账号和邮件服务器信息应该与你实际环境信息保持一致。在本例中，我们的 IMAP 服务器和 SMTP 服务器同为 cesoo.com，IMAP 端口号是默认的 143，SMTP 端口号是

默认的 25，账号用户名为 test1，如图 8-4 所示。

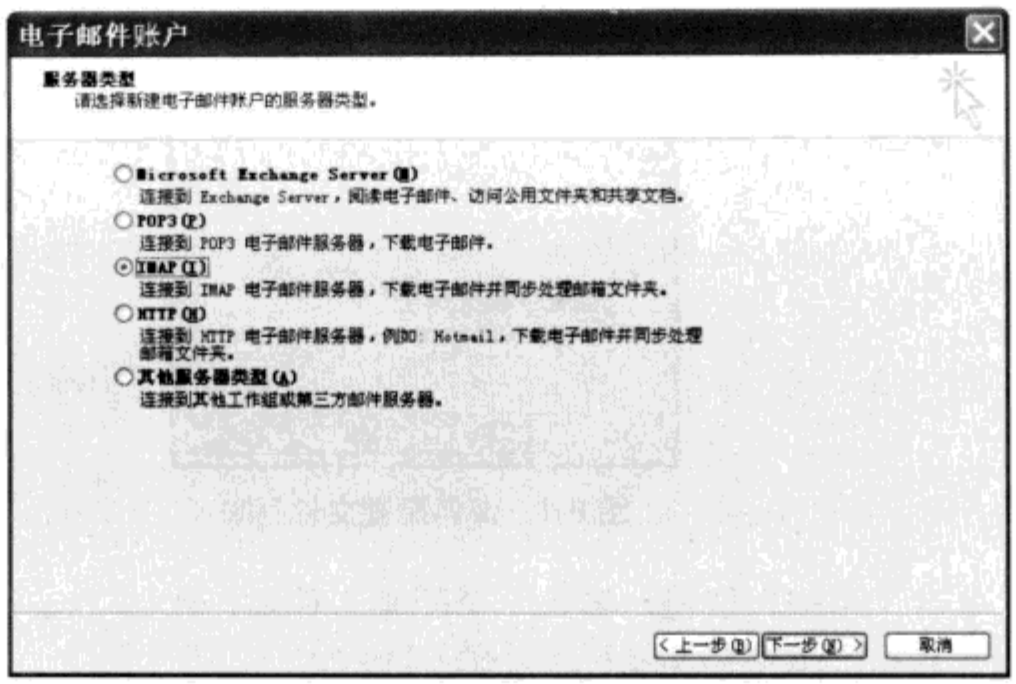


图 8-3　邮件配置向导第二步

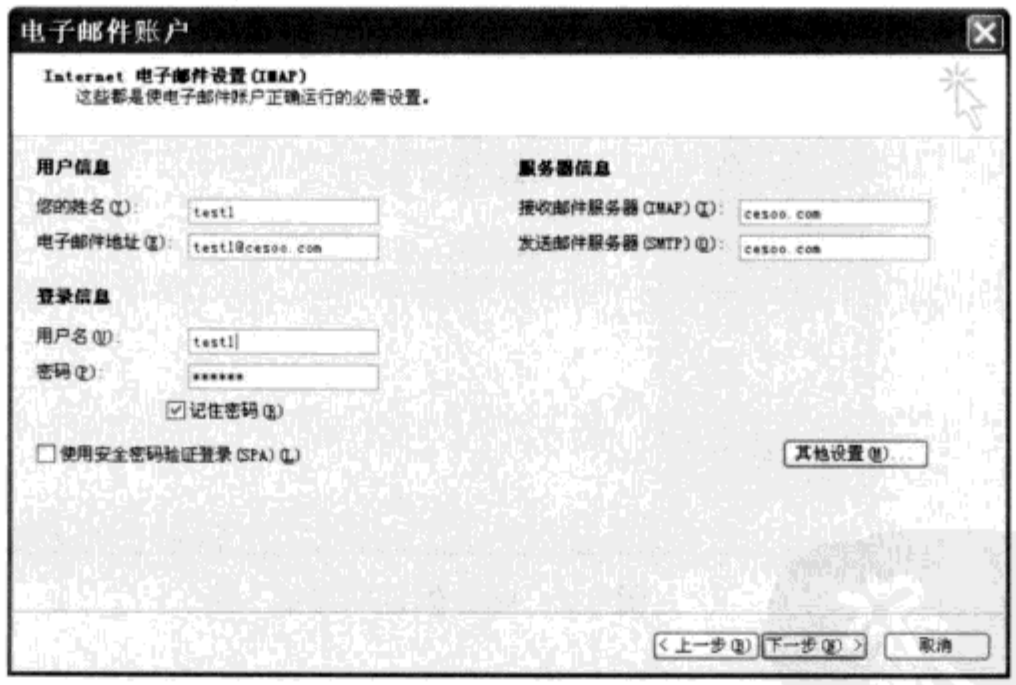


图 8-4　邮件配置详细信息

⑤ 确认输入信息完整后，单击“下一步”按钮，弹出提示成功信息对话框，Outlook 2003 的 IMAP 和 SMTP 配置完成，如图 8-5 所示。

配置文件成功创建后，我们就可以收发邮件了。而 VU 则能把这个过程全部捕捉下来，生成脚本。

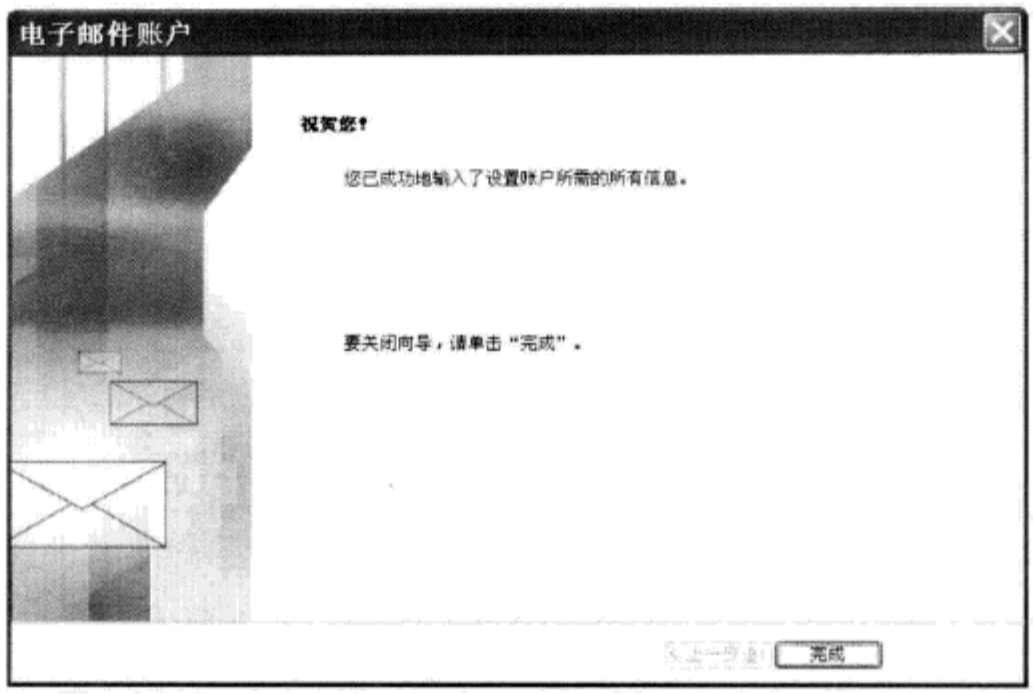


图 8-5　邮件配置成功页面

8.2.1　使用 SMTP Vuser 录制 Outlook 2003 发送邮件

打开 VU，在“File”菜单下，选择“New”，在 Vuser 类型列表中，选中“Simple Mail Protocol（SMTP）”，如图 8-6 所示。

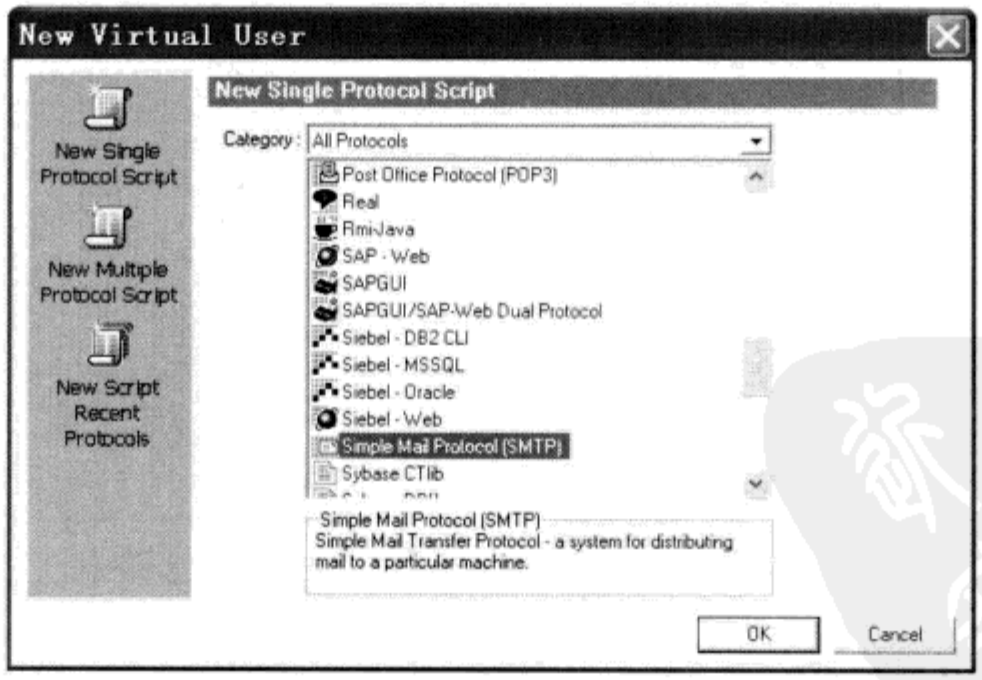


图 8-6　在虚拟用户列表中选择 SMTP Vuser

单击“Record”按钮，会弹出如图 8-7 所示的录制选项设置对话框。

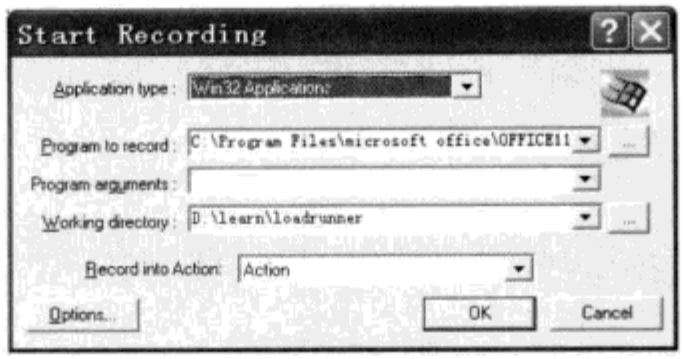


图 8-7　在录制选项中设置 Outlook 应用程序路径

“Application type”（应用程序类型）选择“Win32 Applications”，在“Program to record”（录制程序）选项中输入 Outlook 2003 的应用程序路径，在本例中为“C:\Program Files\microsoft office\OFFICE11\OUTLOOK.EXE”，单击“OK”按钮，开始录制。

录制开始后，VU 会找到应用程序路径，并启动 Outlook 2003。其主界面如图 8-8 所示。

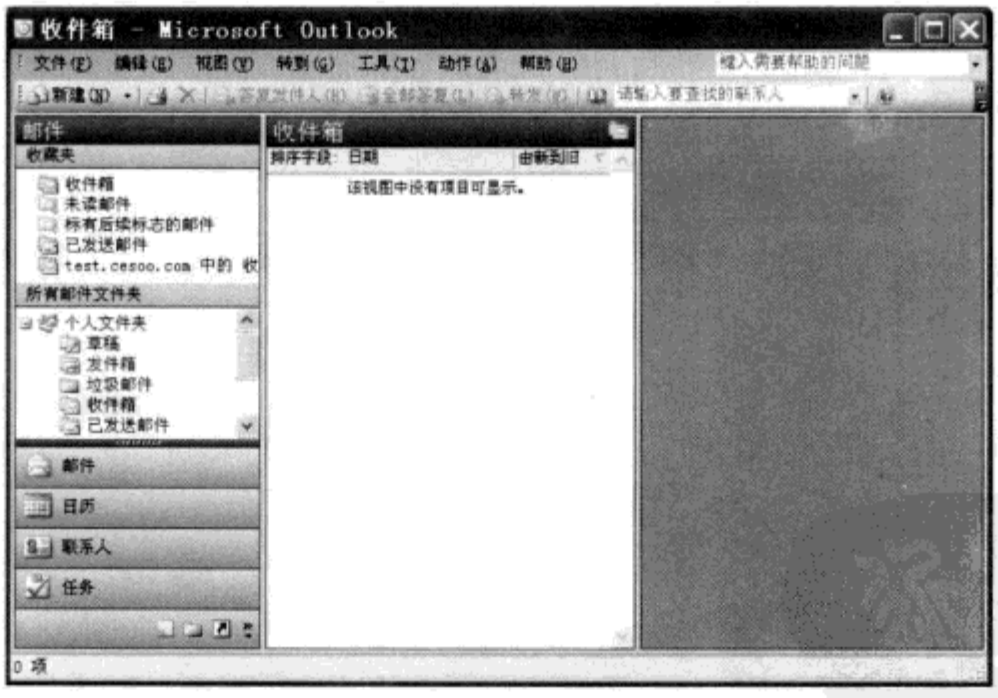


图 8-8　Outlook 启动成功后的主页面

这时 VU 录制工具条有如下提示，如图 8-9 所示。

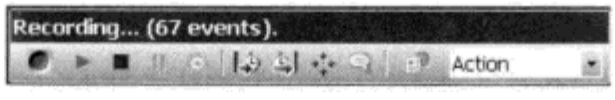


图 8-9　Outlook 登录引起网络 67 次交互事件

这说明 Outlook 2003 完成登录这个操作，在网络上已经有了 67 次交互。

下面我们要重点观察发送邮件的操作，因此要在这里定义一个 Transaction（事务），用来度量发邮件的操作。

单击工具条上的 Transaction 开始点，定义一个名为“sendMail”的 Transaction，如图 8-10 所示。

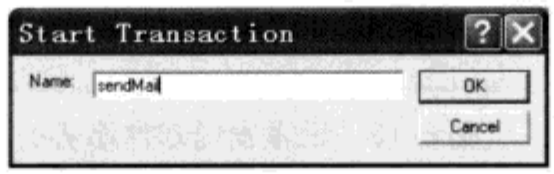


图 8-10　设置发送邮件的事务开始点

回到 Outlook 2003，继续操作。单击“新建”按钮，会弹出新邮件的窗口，在收件人、邮件标题、邮件正文中填写信息，如图 8-11 所示。



图 8-11　在 Outlook 中构建一封新邮件

单击“发送”按钮，邮件被发送。这时我们再观察 VU 录制工具条，发现如图 8-12 所示的提示。



图 8-12　Outlook 发送邮件引发的网络交互事件

交互事件已经由之前的 67 次增长到 91 次，这说明发送邮件的操作在网络上产生了 91–67=24 次交互。

单击工具条上结束 Transaction 的按钮，自动提示“sendMail”，直接单击“OK”按钮，如图 8-13 所示。

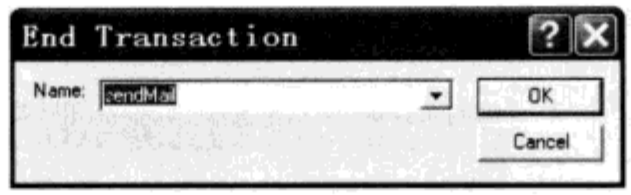


图 8-13　设置发送邮件事务的结束点

停止 VU 录制，VU 生成脚本如下：

```
Action()
{
    lr_start_transaction("sendMail");
    smtp1 = 0;
    smtp_logon_ex(&smtp1, "SmtpLogon",
        "URL=smtp://www.cesoo.com",
        "CommonName=LoadRunner User",
    LAST);

     smtp_send_mail_ex(&smtp1, "SendMail",
        "To=test1@cesoo.com",
        "From=test1@cesoo.com",
        "Subject==?utf-8?B?6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu25qCH6
aKYdGVzdA==?=",
        "ContentType=multipart/alternative;",
         MAILOPTIONS,
         "X-Mailer: Microsoft Office Outlook, Build 11.0.5510",
         "Thread-Index: AciOgCIrErOOfaSTRUulBNIY7vEV0Q==",
        "X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.3028",
        MAILDATA,
       "AttachRawFile=mailnote1_01.dat",
       "AttachRawFile=mailnote1_02.dat",
    LAST);

     smtp_logout_ex(&smtp1);
     smtp_free_ex(&smtp1);
```

```
        lr_end_transaction("sendMail",LR_AUTO);

        return 0;
    }
```

8.2.2　对 SMTP Vuser 录制生成的脚本进行分析

我们在 Outlook 上的一系列操作，包括登录和发送邮件，被 VU 转换成一系列函数，比如 smtp_logon_ex 和 smtp_send_mail_ex 等。下面我们对这些函数进行分析。

（1）在 lr_start_transaction(“sendMail”)语句之前未有其他语句。而在 sendMail 之前，实际上在 Outlook 之前已经完成了登录，并且在网络上有了 67 个事件交互。显然，SMTP Vuser 对这些网络交互事件并不感兴趣，因为它们并没有转化成脚本。这只能说明一个事实：Outlook 登录操作的底层走的网络协议并不是 SMTP 协议（实际上是 IMAP 协议）。

（2）smtp_logon_ex 是 SMTP Vuser 提供的一个函数，实现的是 SMTP 的登录认证。

（3）smtp_send_mail_ex 是 SMTP Vuser 的一个最主要的功能，其作用是将一封邮件发送到指定的 E-mail 地址。

- “From=test1@cesoo.com”是设置发件人地址。
- “To=test1@cesoo.com”是设置收件人地址。
- “Subject==?utf-8?B?6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu25qCH6aKYdGVzdA==?=”是设置邮件的标题（subject）。

从上一节的 SMTP 协议规范我们知道，SMTP 网关一直保持 7 位 ASCII 码的“古老传统”，因此任何多字节的数据在经过 SMTP 协议之前必须要经过编码。在这里，“?utf-8?”说明 Oultlook 2003 用的是 utf-8 编码方式，而“6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu25qCH6aKYdGVzdA”则是 “这是一封测试邮件的标题”的编码后字节表现形式。

（4）“AttachRawFile=mailnote1_01.dat”和“AttachRawFile=mailnote1_ 02.dat”，则是邮件正文的传送方式。检查录制后的脚本，发现左侧导航栏中多了“mailnote1_01.dat”和“mailnote1_02.dat”两个节点，如图 8-14 所示。

点击打开 mailnote1_01.dat，我们看到如下数据：

```
Content-Type: text/plain;
charset="utf-8"
Content-Transfer-Encoding: base64
6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu255qE5q2j5paHdGVzdA0K
```

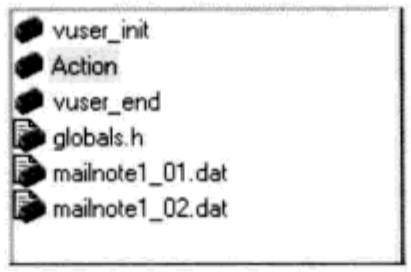


图 8-14 SMTP 虚拟用户脚本中的数据区

这是对邮件正文的编码，同样的 utf-8 编码，“6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu255qE5q2j5paHdGVzdA0K”则是“这是一封测试邮件的正文”的 utf 编码后字节表现形式。

点击打开 mailnote1_02.dat，我们看到如下数据：

```
Content-Type: text/html;
charset="utf-8"
Content-Transfer-Encoding: quoted-printable
=EF=BB=BF<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional
//EN">
<HTML><HEAD>
<META http-equiv=3DContent-Type content=3D"text/html; charset=
3Dutf-8">
<META content=3D"MSHTML 6.00.2900.3157" name=3DGENERATOR></HEAD>
<BODY>
<DIV><FONT face=3D=E5=AE=8B=E4=BD=93 =
size=3D2>=E8=BF=99=E6=98=AF=E4=B8=80=E5=B0=81=E6=B5=8B=E8=AF=9
5=E9=82=AE=E4=
=BB=B6=E7=9A=84=E6=AD=A3=E6=96=87test</FONT></DIV></BODY></HTML>
```

这些看起来都不是很陌生，和网页的 DOM 源码非常相像。没错，因为 Outlook 发送邮件的格式默认是 html 的，所以其实邮件正文里暗含了 html 的格式数据。

8.2.3　回放 SMTP Vuser 脚本并分析网络日志

我们通过上一节的分析已经得知，SMTP Vuser 将发送邮件的动作转化成了 VU 函数，但这是不是对 SMTP 协议一个完整的模拟呢？我们知道 SMTP 协议是由一系列操作原语组成的，如 HELO、MAIL FROM、RCPT TO 等。如果回放 SMTP Vuser 脚本，同时我们在网络上能捕获到这些原语，就说明 SMTP Vuser 确实是对 SMTP 协议进行了封装和实现。

回放脚本，同时使用 Ethereal 在网络上捕捉从客户端到 cesoo 服务器 25 端口的数据。我们捕捉到网络交互如下：

```
     发送者              接收者              数据内容
01. Intel_a3:e8:ef BroadCast           Who has 192.168.1.100?
02. 1d:7d:4a:66    Intel_a3:e8:ef      192.168.1.100 is at 1d:7d:4a:66
03. 192.168.1.1    192.168.1.100       smtp syn
04. 192.168.1.100  192.168.1.1         220 server ready.
05. 192.168.1.1    192.168.1.100       HELO
06. 192.168.1.100  192.168.1.1         cesoo.com hello, glad to meet u
07. 192.168.1.1    192.168.1.100       MAIL FROM:test1@cesoo.com
08. 192.168.1.100  192.168.1.1         250 sender is OK
09. 192.168.1.1    192.168.1.100       RCPT TO:test1@cesoo.com
10. 192.168.1.100  192.168.1.1         250 receipt is OK
11. 192.168.1.1    192.168.1.100       DATA
12. 192.168.1.100  192.168.1.1         Enter mail, end with "."
13. 192.168.1.1    192.168.1.100       Message body
14. 192.168.1.1    192.168.1.100       Message body
15. 192.168.1.1    192.168.1.100       Message body
16. 192.168.1.100  192.168.1.1         221 cesoo.com closing connect
```

注：以上数据经过过滤和整理。192.168.1.1 为客户端，192.168.1.100 为 Server 端，即 cesoo.com。

我们可以看到 1 和 2 是网络 ARP 协议，用来确定服务器地址。而下面的交互

完全遵循 SMTP 协议规范，老老实实地按照 HELO、MAIL FROM、RCPT TO、DATA 的顺序进行对话。所以我们从这里可以看到 SMTP Vuser 确实对 SMTP 协议进行了一次完全的封装。最后在网络中传递的 message body 如下：

```
    Message-ID: <000047e9148d$00000001$00bb6f3c@cesoo.com>
    From: "LoadRunner User" <test1@cesoo.com>
    Date: 星期二, 25 三月 2008 23:04:45 +08:00
    Subject:
=?utf-8?B?6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu25qCH6aKYdGVzdA==?=
    MIME-Version: 1.0
    Content-Type:multipart/alternative;
                    boundary="------=_NextPart_00000001_000009f8
.00bb6f3c"
    X-Mailer: Microsoft Office Outlook, Build 11.0.5510
    Thread-Index: AciOghaui9AQdpK8TO+CcxyY5WXVPQ==
    X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.3028
    This is a multi-part message in MIME format.
    --------=_NextPart_00000001_000009f8.00bb6f3c
    Content-Type: text/plain;
    charset="utf-8"
    Content-Transfer-Encoding: base64

    6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu255qE5q2j5paHdGVzdA0K

    --------=_NextPart_00000001_000009f8.00bb6f3c
    Content-Type: text/html;
                    charset="utf-8"
    Content-Transfer-Encoding: quoted-printable

    =EF=BB=BF<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional
//EN">
    <HTML><HEAD>
    <META http-equiv=3DContent-Type content=3D"text/html; charset=
3Dutf-8">
    <META content=3D"MSHTML 6.00.2900.3157" name=3DGENERATOR></HEAD>
    <BODY>
    <DIV><FONT face=3D=E5=AE=8B=E4=BD=93=
    size=3D2>=E8=BF=99=E6=98=AF=E4=B8=80=E5=B0=81=E6=B5=8B=E8=AF=9
5=E9=82=AE=E4=
    =BB=B6=E7=9A=84=E6=AD=A3=</FONT></DIV></BODY></HTML>
```

以上就是网络中 SMTP 协议实际传输的数据。最后我们在 Outlook 端看到的这封邮件，其实是由 Outlook 客户端进行解释并显示的，如图 8-15 所示。

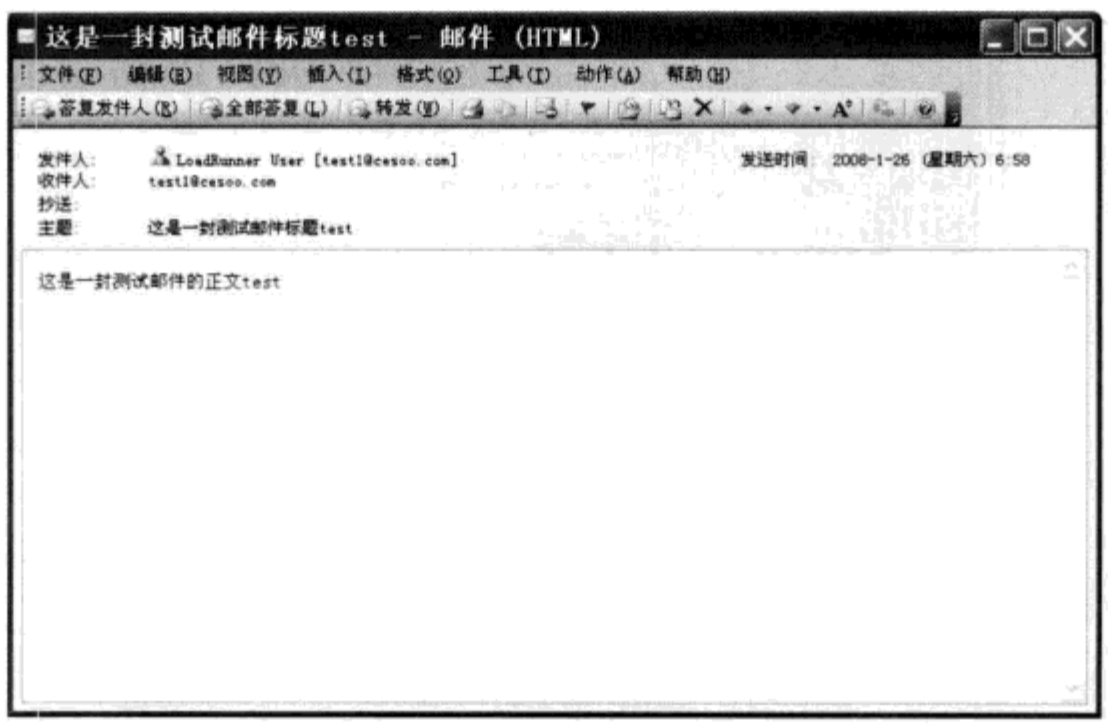


图 8-15　SMTP Vuser 成功回放脚本并发送邮件

8.3　VU Java Vuser 自开发 SMTP 程序

从上节的学习可知，VU 可以使用 SMTP Vuser 来录制生成基于 SMTP 协议的脚本，这对于刚入门的测试工程师来说非常快捷和高效，但同时也为调试脚本带来一定困难，尤其是邮件中包含一些非英文字符，在 dat 文件中都是编码后的字节，难以维护和修改。在这种情况下，如果性能测试工程师具有一定的开发经验和技术，就可以考虑使用 Java Vuser 自主开发 Java 程序来达到相同的目的。

8.3.1　VU 创建 Java 模板虚拟用户

我们首先需要创建 Java Vuser，然后在 Java Vuser 环境下加载或开发 Java 程序。

启动 VU，在“File”菜单下选择“New”，在弹出的 Vuser 列表中选择“Java

Vuser”，如图 8-16 所示。

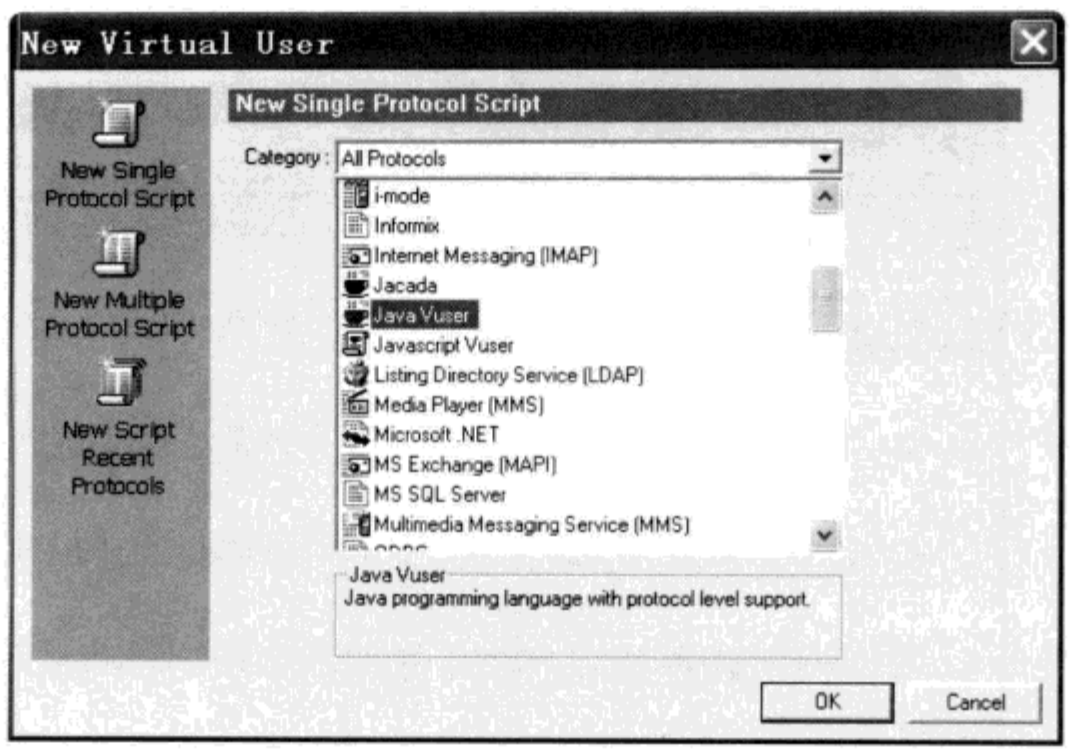


图 8-16 在虚拟用户列表中选择“Java Vuser”

单击“OK”按钮，在 VU 脚本中我们看到生成如下模板脚本：

```
import lrapi.lr;
public class Actions
{

    public int init() {
    return 0;
    }//end of init


    public int action() {
    return 0;
    }//end of action


    public int end() {
    return 0;
    }//end of end
}
```

由上可见，Java Vuser 中的 Int()、action()、end()函数和 C Vuser 中的 vuser_init、action、vuser_end 一样，迭代的设置只对 action 生效，而对 init 和 end 无效。我们的开发主要在 action 函数中完成。

8.3.2　设置 Java 环境

我们知道 Java 环境最主要的参数有两个：一个是 Java 虚拟机 path，另一个是 classpath。

打开 Vuser 的运行时设置（Run-time Settings）窗口，选择 Java VM，如图 8-17 所示。

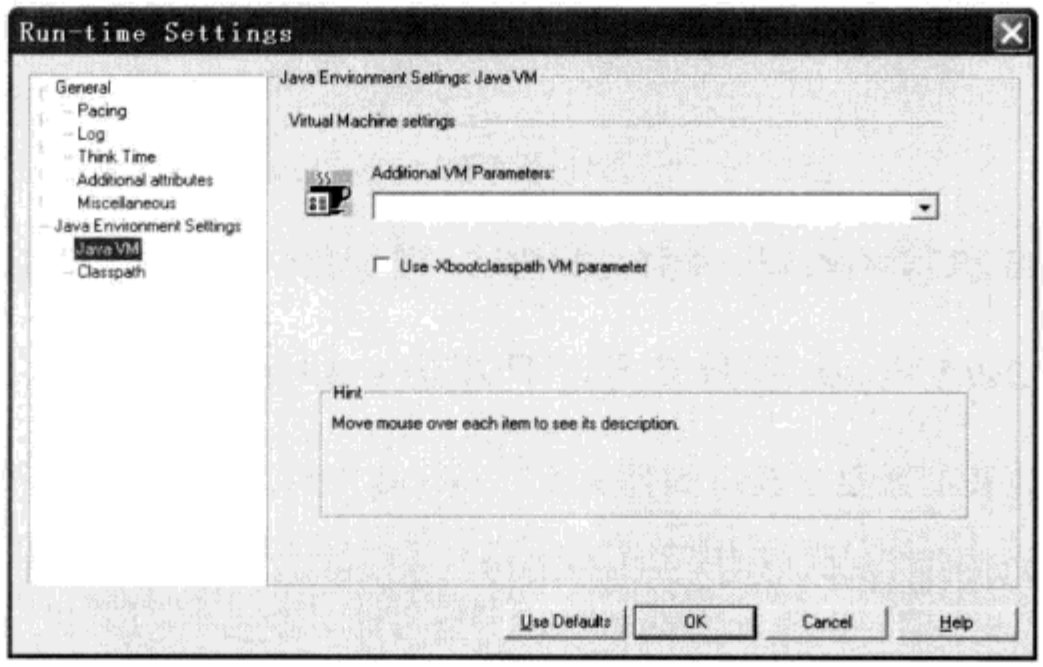


图 8-17　运行时设置中的 Java VM 配置页面

这里如果 Xbootclasspath 没有特殊的参数设置，对此页面不要做任何修改。因为 VU 将自动从 path 中获得 Java 虚拟机的路径。

单击“Classpath”，显示如图 8-18 所示的配置页面。

对于 SMTP 协议，Java 开源组织已经提供了很多相关 package 封装，我们可以在 Internet 上下载，比如在 jakarta 网站。

我们在这里添加 3 个 mail 的 jar 包，即 mail.jar、poi.jar 和 jakarta-oro-2.0.8.jar。单击“OK”按钮，关闭“Run-time Settings”窗口。

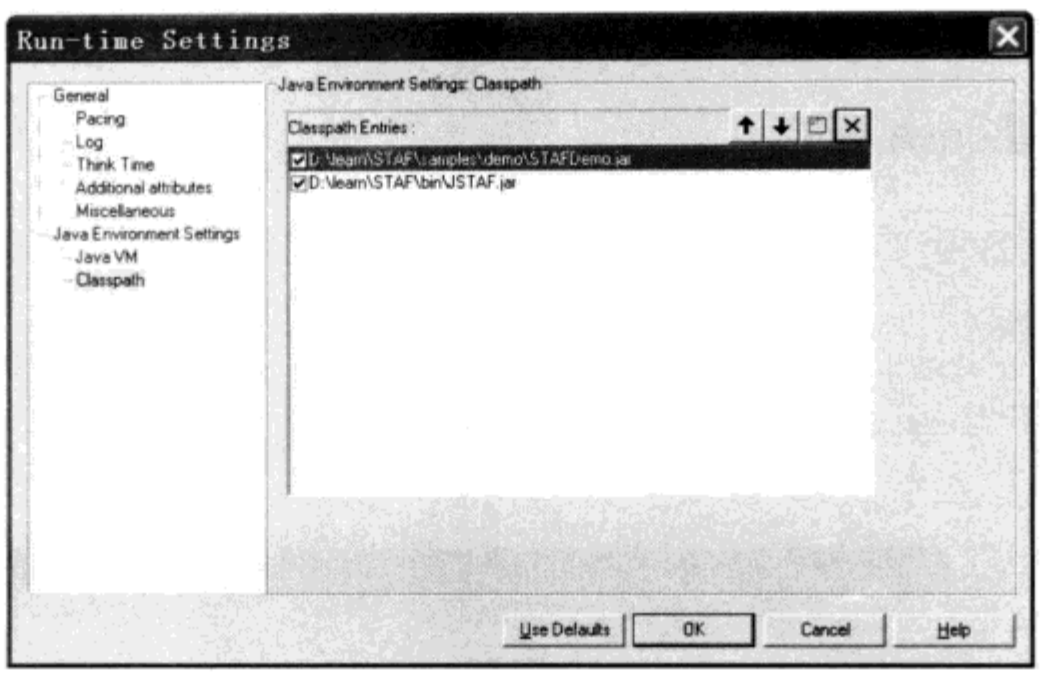


图 8-18　运行时设置中的 Java Classpath 配置页面

单击 VU 运行按钮，编译运行通过，验证我们的设置是正确的。

8.3.3　在 Java Vuser 中开发 SMTP 发送 mail 脚本

首先我们需要在 Java 文件头进行 import 声明，包含相关 package。

```
import lrapi.lr;
import java.io.File;
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.Date;
import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;
import org.apache.poi.hssf.usermodel.*;
import com.sun.mail.smtp.*;
```

再次运行脚本，以验证 import 是否成功，否则 Java 会报错，显示 package 找不到。

在 action 函数中编写发送邮件的 Java 代码，如下：

```
public int action() {

    boolean ssl = false;
    //获得 SMTP 环境
    Properties props = System.getProperties();
    //设置 SMTP 主机地址
    props.put("mail.smtp.host", "192.168.1.100");
    //设置 SMTP 端口号
    props.put("mail.smtp.port","25");
    //设置 SMTP 用户名
    props.put("mail.smtp.user","test1");
    Session session=Session.getInstance(System.getProperties(),null);
    //设置邮件 header 字段
    String mailer = "send from VU java SMTP";
    //设置收件人
    String sendTo = "test1@cesoo.com";
    //设置抄送人
    String sendCC = "test1@cesoo.com";
    //设置发件人
    String sendFrom = "test1@cesoo.com";
    try{
       //获得发送实体
       SMTPTransport t = (SMTPTransport)session.getTransport(ssl ?
"smtps" : "smtp");
       //与 SMTP 主机相连
       t.connect("192.168.1.100","test1","123456");
       MimeMessage msg = new MimeMessage(session);
       //设置邮件各个字段
       String subject = "testing mail subject";
       String personalName = "LR JAVA Vuser";
       String body = "testing mail body";
       String charset = "utf-8";
       InternetAddress[] to = new InternetAddress[1];
       InternetAddress from = new
    InternetAddress(sendFrom,personalName,charset);
       to[0] =new InternetAddress(sendTo,personalName,charset);
       msg.setFrom(from);
       msg.setRecipients(Message.RecipientType.TO,to);
```

```
            msg.setSubject(subject,charset);
            msg.setText(body,charset);
            msg.setSentDate(new Date());
            //发送邮件
            t.sendMessage(msg, msg.getAllRecipients());
        }
        catch (Exception e){
            e.printStackTrace();
        }
        return 0;
    }//end of action
```

以上代码运行成功后，Outlook 即可收到一封邮件，如图 8-19 所示。

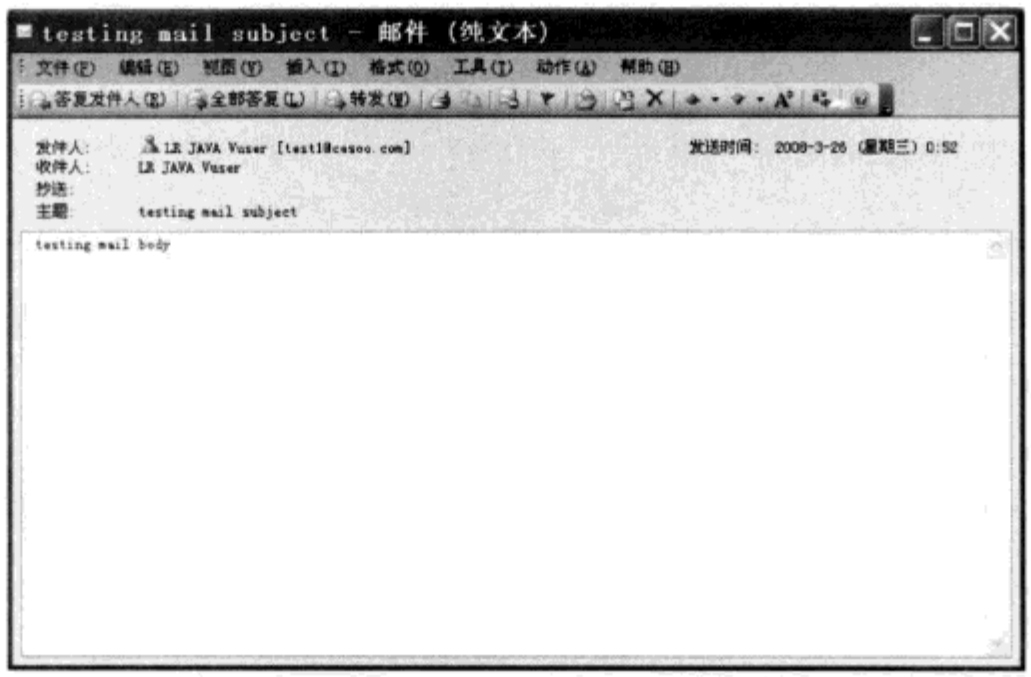


图 8-19 Java Vuser 成功回放脚本并发送邮件

8.3.4 参数化增强脚本

以上脚本中的配置信息和邮件字段信息可使用参数化方法，转移到参数表中存储。参数化后，代码有如下变化：

```
    public int action() {

        boolean ssl = false;
         //获得 SMTP 环境
```

```
        Properties props = System.getProperties();
        //设置 SMTP 主机地址
        props.put("mail.smtp.host", "<SMTPHost>");
        //设置 SMTP 端口号
        props.put("mail.smtp.port","<SMTPPort>");
        //设置 SMTP 用户名
        props.put("mail.smtp.user","<UserName>");
        Session
session=Session.getInstance(System.getProperties(),null);
        //设置邮件 header 字段
        String mailer = "<MailHeadere>";
        //设置收件人
        String sendTo = "<SendTo>";
        //设置抄送人
        String sendCC = "<sendCC>";
        //设置发件人
        String sendFrom = "<sendFrom>";
        try{
           //获得发送实体
           SMTPTransport t = (SMTPTransport)session.getTransport(ssl ?
"smtps" : "smtp");
           //与 SMTP 主机相连
           t.connect("<SMTPHost>","<UserName>","<Passwd>");
           MimeMessage msg = new MimeMessage(session);
           //设置邮件各个字段
           String subject = "<mailSubject>";
           String personalName = "<PersonalName>";
           String body = "<mailBody>";
           String charset = "utf-8";
           InternetAddress[] to = new InternetAddress[1];
           InternetAddress from = new
      InternetAddress(sendFrom,personalName,charset);
           to[0] =new InternetAddress(sendTo,personalName,charset);
           msg.setFrom(from);
           msg.setRecipients(Message.RecipientType.TO,to);
           msg.setSubject(subject,charset);
           msg.setText(body,charset);
           msg.setSentDate(new Date());
           //发送邮件
           t.sendMessage(msg, msg.getAllRecipients());
```

```
        }
      catch (Exception e){
         e.printStackTrace();
      }

       return 0;
}//end of action
```

参数化后，验证脚本，能够通过。

我们就可以在参数表中更改测试数据，发送不同字符的邮件了！

中文邮件：如图 8-20 所示。

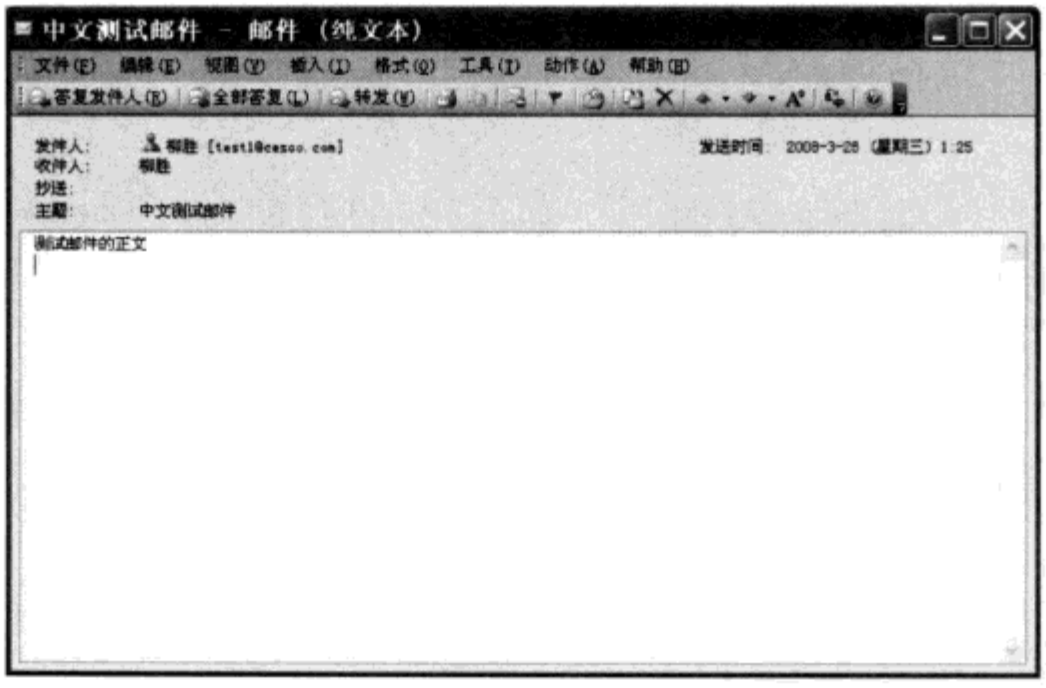


图 8-20　Java Vuser 参数化后成功发送中文邮件

日文邮件：如图 8-21 所示。

由此可见，通过 Java Vuser 的自开发程序代码，脚本的灵活性大大增强。这是录制脚本无法比拟的优势。

录制和开发是 VU 生成脚本的两种方式，我们应该根据项目需求、时间、资源等综合因素考虑选择最合适的方式。

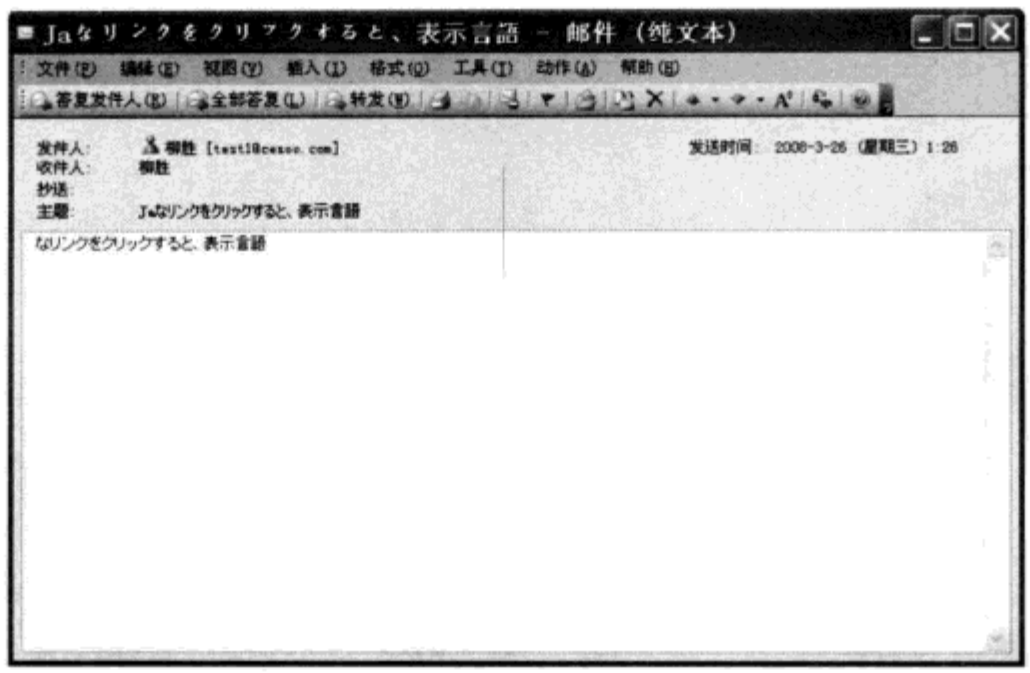


图 8-21　Java Vuser 参数化后成功发送日文邮件



第9章

学以致用——一步一步做Web系统性能测试

本章我们将通过一个性能测试实例，结合 GAME(A)模型，来介绍如何使用 LoadRunner 做性能测试。

9.1 软件系统背景及架构设计介绍

本系统应我国金融信息化发展设计，采用当今比较先进和流行的技术，是应用在城域网上的大型分布式应用系统。

本系统为金融交易系统，就运行模式而言，实现会员制模式，所有参加交易的单位均首先成为交易中心的会员；所有的交易会员可以委托交易员执行交

易指令，一个交易会员可以有一个（或多个）首席交易员和一个（或多个）交易员。

整个系统分为 3 个层次：交易中心、分中心、交易会员（见图 9-1）。

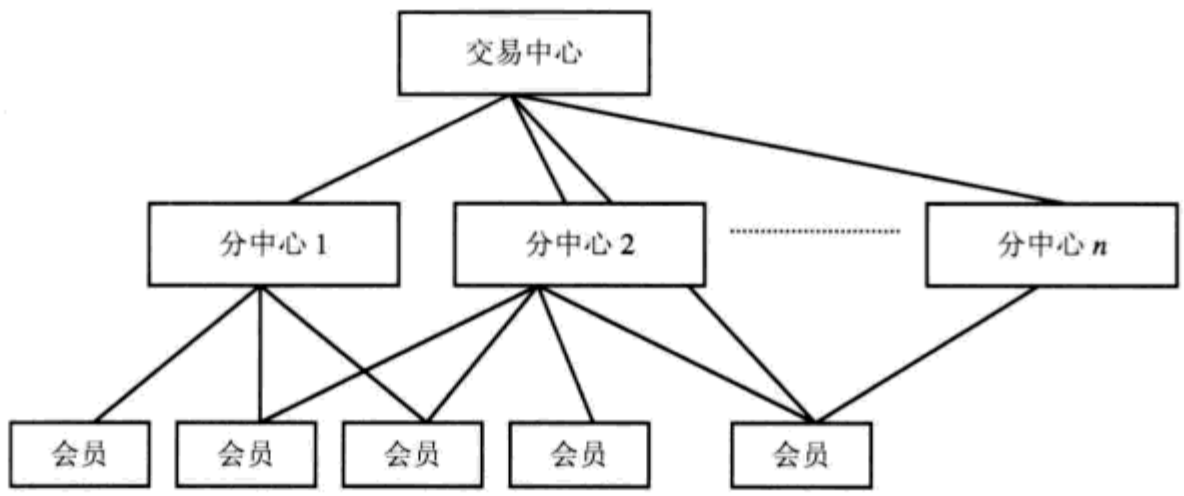


图 9-1　被测金融系统运行模式图

作为交易的撮合和数据集散中心，交易中心是系统的核心所在，其担负着系统的主要功能，如交易撮合、信息查询等。

分中心担负着系统的中介功能，为会员和交易中心之间构建了一个桥梁，同时还担负着部分查询功能，比如同一个分中心管辖下的会员进行询价时，可以只在分中心进行处理，而不必上行到交易中心。

会员是整个体系的交易主体，其可以连接到分中心和交易中心进行交易。

本系统遵循 J2EE 规范，采用 B/S 体系结构进行设计和开发，是一个建立在广域网上的分布式应用系统。业务主要分为交易业务和查询业务，查询业务采用 J2EE 规范，交易业务以 J2EE 体系架构为基础，进行进一步的处理，采用了 TCP 的四层结构。系统体系结构图如图 9-2 所示。

采用中间件技术构建三层（或多层）应用结构已经成为应用开发和运行的主流技术，其核心概念是利用中间件将应用的表示逻辑（客户界面）、业务逻辑（服

务组件）和数据管理（数据库）分为三个不同的处理层。其中业务逻辑作为中间层实现核心业务逻辑服务，中间件管理各种组件，接收客户的服务请求，向数据库服务器提交数据操作，并将处理结果返回给请求者。

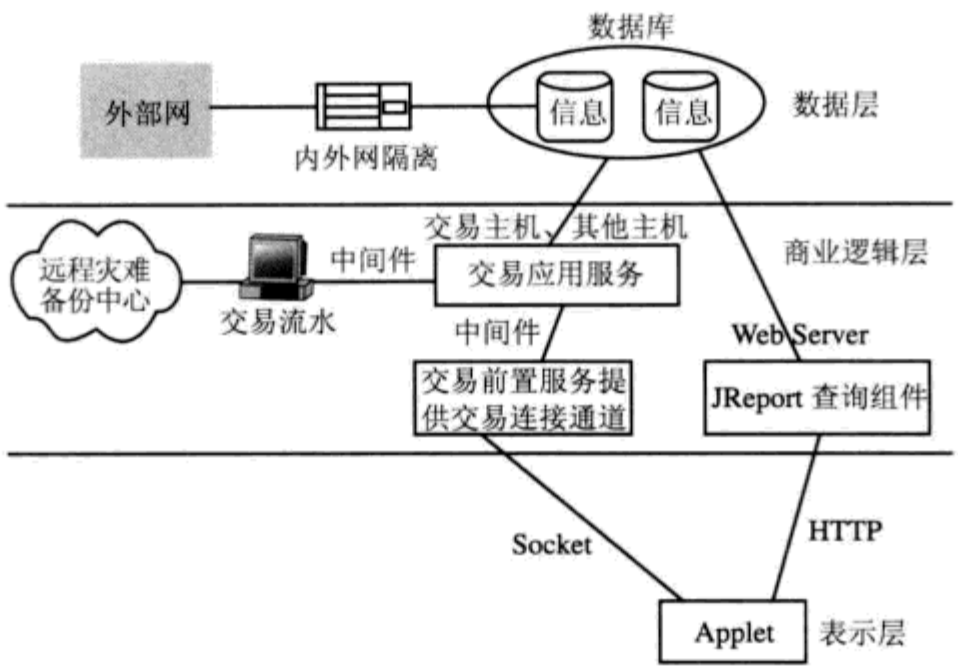


图 9-2 被测系统体系结构图

- 表示层提供协议控制和用户界面，与系统最终用户实现直接交互。负责接收用户的服务请求，通过 Socket 连接向交易前置服务发送。
- 商业逻辑层作为中间层实现核心业务逻辑服务，这些组件由中间件管理，接收客户的服务请求，向交易主机提交数据操作，并将交易主机的业务处理结果返回给请求者。
- 数据层负责整个系统中数据信息的存储、访问及其优化。

通过使用中间层，实现了业务逻辑与表示逻辑、业务逻辑与数据管理的分离，使得系统能够灵活地适应用户业务逻辑的变化。

在物理上，本系统的结构图如图 9-3 所示。

- 数据库主机：数据库主机硬件采用 IBM P660-6M1 小型机，操作系统是 AIX 4.3.3，运行 DB2 数据库服务程序。通过 DB2 C API 与交易主机系统通信，JDBC 与查询前置机通信。

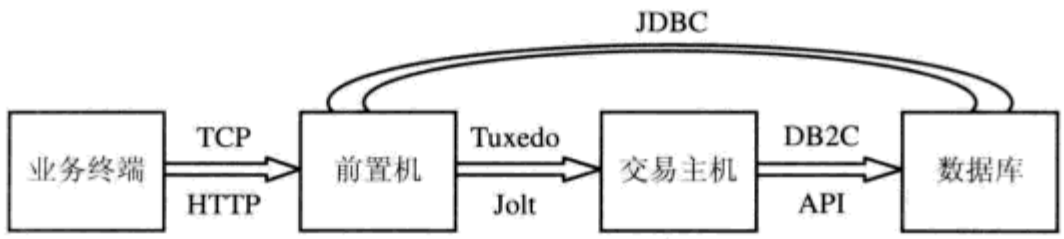


图 9-3　被测系统物理结构图

- 交易主机：数据库主机硬件采用 IBM P660-6M1 小型机，操作系统是 AIX 4.3.3，运行 Tuxedo Jolt 中间件与交易前置机连接，通过 DB2 C API 与数据库连接。在 Tuxedo 中间件上运行业务处理程序，按交易规则处理前置机发来的交易指令。
- 前置机：前置机硬件采用 IBM 640 小型机，操作系统是 AIX 4.3.3。前置机上运行交易前置机和查询前置机，它通过 TCP/HTTP 与业务终端通信。交易前置机通过 Tuxedo Jolt 客户端与交易主机通信。查询前置机运行在 WebLogic 应用服务器上并运行 JReport 组件，完成对查询流指令的发送并接收数据库返回的结果给业务终端。交易前置机运行服务程序，接收终端请求报文并将其转发给交易主机。并且通过轮询和同步反馈接收交易主机返回的报文，将其转发给业务终端。
- 终端机：通过 TCP/HTTP 与前置机系统通信。运行 Applet 程序，向前置机发送请求报文，并接收前置机返回的回应报文。

9.2　性能要求和性能指标分析（Goal 阶段）

9.2.1　性能指标

用户对本系统要求的性能指标如表 9-1 所示。

下面根据此系统和给定的性能指标来进行性能测试。

表 9-1 用户要求性能指标表

指标种类		用户需求
登录		交易主机能够处理 750 用户/分钟，至少支持上百用户并发登录响应时间不超过 30 秒
业务	交易	交易主机能够处理 300 笔/分钟，响应时间不超过 10 秒
	查询	交易主机能够处理 400 笔/分钟，响应时间不超过 10 秒

9.2.2 业务模型分析和需求细化

经过和用户的讨论，双方对性能需求又得到进一步的明确：

（1）上述性能指标是经过业务模型计算而得出的。可以确信在此性能指标满足的情况下，系统上线不会有性能问题。

（2）此软件系统具有 5000 个来自不同网络的客户终端，有局域网、Modem 拨号等，其中大多采用 Modem 拨号。这个事实对于性能测试有两个显而易见的推论：

- 性能测试时，数据库中应该至少已经存在 5000 个用户相关记录。
- LoadRunner 应该为每个虚拟用户分配 Modem 带宽，而不是默认的最大带宽。

（3）计划使用 5 年。性能测试时，数据库中要有一定数量的历史数据，包括各种交易记录。

9.3 性能测试方案和用例设计（Analysis 阶段）

性能测试的目的是最大程度地模拟真实业务场景，来验证系统的性能指标，并发现可能存在的性能瓶颈。

通过对本系统的分析，我们可以看到本系统大体上由终端、前置机、交易主机、数据库主机等节点组成。

在整个业务流程中，业务终端→前置机→交易主机→数据库主机形成了一个压力流串，每个节点在压力下能够正常工作是整个系统正常运转的基础。也就是说，如果其中任意一个节点在业务压力下发生了拥塞、处理不力等不正常情况，那么整个系统都无法正常运转。

下面我们再对系统压力流进行细致地分析，以得出我们本次性能测试的实施方案。

首先，从交易终端到前置机，交易终端产生业务报文发送至前置机，前置机上运行查询前置机和交易前置机，查询前置机通过 JDBC 直接与数据库系统进行通信，前置查询机运行在 WebLogic 应用服务器上并运行 JReport 组件，完成对查询指令的发送并接收数据库返回的结果给终端界面系统。交易前置机通过 Tuxedo Jolt 客户端与交易主机进行通信。交易前置机运行服务程序，接收终端协议报文并按协议进行处理后，根据不同的业务类型分别通过交易流/私有流/市场流将指令发给交易主机，并通过轮询和同步反馈接收交易主机返回的报文，将其返回给终端界面系统。

由以上分析，我们可以整理出整个系统的两条压力流程线，之所以我们把其分为两条流程线，是因为交易前置机和查询前置机的工作原理完全不同，下与交易终端的连接，上与交易主机的连接也完全是独立的两个通路。

- 终端→交易前置机→交易主机→数据库系统
- 终端→查询前置机→数据库系统

下面我们先独立分析两条流程线，之后将再次综合分析，以考虑二者之间的相互影响作用。

第一条路线上主要运行的是交易指令，以及交易指令所产生的反馈信息。

当系统运作时，千百个交易终端指定交易对手，并把业务交易操作的压力施加到前置机，然后前置机通过开发方编写的服务程序接收到交易指令，并检验其

合法性，再通过交易中间件 Tuxedo 的客户端把业务的压力传递给交易主机进行处理。交易主机进行必要的金融计算和业务逻辑运行，得出反馈结果，一方面顺原路返回到发起交易的交易终端上去，另一方面把交易的信息传送到接收交易的交易对手中去，分析如图 9-4 所示。

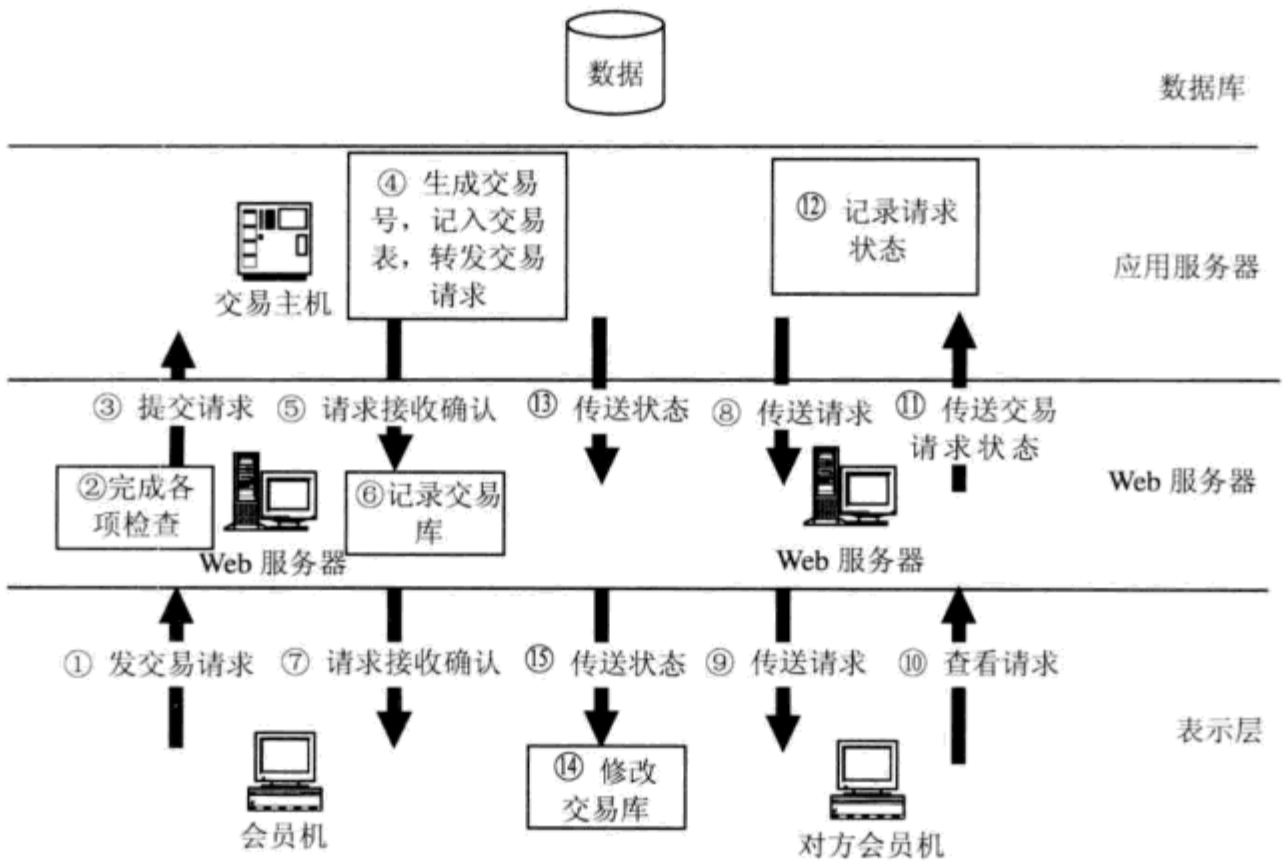


图 9-4　金融业务流程图

流程如下：

① 发出交易请求。由交易一方会员通过会员端机器向 Web 服务器发出交易请求。

② 完成检查。在 Web 服务层，首先进行各项检查，包括交易请求的有效性、请求者的身份合法性等。对不合格的交易请求，则拒绝继续向交易主机报价，并向交易请求者发送处理结果。

③ 提交交易请求。通过检查后的交易请求由 Web 服务器向交易主机进行提交，同时记录日志。

④ 主机处理。交易主机接收 Web 服务器请求的交易，进行处理，如生成交易号、记入交易表、交易请求转发等。

⑤ 交易请求接收确认。交易主机将处理结果反馈给请求的 Web 服务器。

⑥ 记录交易请求库。Web 服务器接收到交易主机的反馈信息后，记录本地交易请求库，以备查询。

⑦ 交易请求接收确认。Web 服务器将报价接收确认转发给会员机。

在交易主机将处理结果反馈给请求的 Web 服务器时，同时还要将交易请求发送给交易的另外一方。流程如下：

⑧ 传送交易请求。交易主机获得交易对方的 Web 服务器信息，将交易请求发送给对应的 Web 服务器。

⑨ Web 服务器继续将交易请求信息发送给对方会员。

⑩ 对方会员查看交易请求，系统将该状态向 Web 服器提交。系统记录会员是否查看了交易请求。

⑪ Web 服务器将状态向应用服务层传送。

⑫ 应用服务层对相应的交易进行状态记录。

⑬ 交易成功后，应用服务器向所有的 Web 服务器发送交易成功的消息。

⑭ 接收到消息的 Web 服务器进行消息转发。

⑮ WEB 服务器继续向会员终端发送状态，告知交易成功的消息。

从整个流程来看，从交易终端发起的交易业务压力能贯穿整个流程串，并施加到各个节点，以收到的反馈结果和反馈时间来考证系统的性能指标。

在本条流程线上的加压主要考验前置机服务程序的多连接承受能力，交易中间件的即时响应能力，交易主机的计算能力，以及数据库的加锁机制。

第二条路线上主要运行的是查询指令。

查询指令产生时，通过 HTTP 协议调用 WebLogic 应用服务器上的相应组件，以 JDBC 接口访问后台的 DB2 数据库，并把数据库返回的结果发送至终端界面。

同流程一，我们同样可以在客户端发起查询业务的压力，来考验 WebLogic 服务器的承受能力，相应业务逻辑的计算能力和数据库的加锁机制。

两条流程线相对独立，但又是互相依赖的。由于是对同一个数据库系统进行读操作和写操作，查询流程的结果依赖于交易流程数据的产生，交易流程产生的数据又通过查询流程得到验证。在进行压力测试时，两者的协同会对数据库形成压力的冲击。

鉴于以上分析，针对各个流程可能出现的薄弱环节，我们决定把本次性能测试分为如下几个子测试来进行。

（1）并发测试

在业务终端模拟多个用户进行并发登录，通过服务器的返回信息、响应时间和服务器端的性能行为表现来考证系统的并发处理能力。

（2）负载测试

此下又有三个子测试。

- 交易流程测试：在交易终端上，通过模拟多个用户，产生大量请求业务交易报文，直接施压在前置机上。本流程测试主要考证交易流程上各个节点在并发和持续大批量业务请求情况下的处理能力。
- 查询流程测试：在查询终端上，通过模拟多个用户，产生大量请求查询的

报文，直接施压在查询前置机上。查询成功与否以是否得到所请求的 Web 页面为标准。本流程测试主要考证查询流程上各个节点在持续大批量业务请求情况下的处理能力。

- 综合测试：在上面两种测试都通过的情况下，进行综合测试，即同时在交易终端和查询终端上发起大量请求，分别施压在交易前置机和查询前置机上。本流程测试主要考证数据库的并发处理读操作和写操作的能力。

9.4　性能测试各种度量的建立（Metrics 阶段）

9.4.1　性能脚本的生成

本次压力测试采用 MI 公司的 LoadRunner 工具，脚本编辑和编译工作在 VU Generator 中进行。

理想的脚本是对现实世界的业务行为进行了完全无误的模拟，这其实是不可能的。我们的目标是使模拟的误差在我们认可的范围之内，并能有方法加以控制。

1．确定脚本生成策略

针对并发登录测试和交易流程测试，由于两者运行机理相同，都是终端调用 Socket Client 和交易前置的 Socket Server 建立连接，将请求消息发送至交易前置机。我们考虑采用将此部分 Java Socket 程序编入测试脚本程序，生成登录和交易业务脚本，通过 LoadRunner 来执行。这样做的好处是绕过终端 IE 界面复杂的处理逻辑，直接施压在前置机上（这种方式同时也带来了偏差，在执行测试场景时通过其他方法得到了一定的弥补）。

2．完善脚本

脚本除了要实现与前置机的 Socket 连接、业务发送等功能，还要建立用户信

息数据池，设置检测点、异常退出点，为脚本执行后的结果统计和分析提供正确的依据。

（1）设置 Transaction。通过用户的性能需求，我们要获得的是登录时间、交易时间和查询时间。因此在脚本中至少设 3 个相应的 Transaction，分别命名为 login、trade 和 query。

（2）参数化数据，包括 username、passwd 和各种报文消息内容等数据，通过 Data Wizard 从数据库直接导入。

（3）设置检查点，一笔交易成功的判断依据是成功地发送出去了交易请求，并且在规定时间内从服务器收到了交易成功的回应。因此，我们在脚本里要对服务器的回应进行分析，报出不同的状态。

针对查询流程测试，由于 LoadRunner 工具支持基于 HTTP 的 Web 访问录制功能，我们将考虑采用以录制脚本为主，手工编写脚本为辅的方法，生成查询业务脚本，通过 LoadRunner 来执行。由于查询脚本基本由录制生成 HTTP 请求和应答，不同的压力测试工具录制会有差别，这里就不再写出查询脚本样例了。

9.4.2 定义用户行为

业务之间存在时间上的顺序关系，交易业务只有在登录完成之后才可以发生，而查询业务主要是查询已交易业务的内容。所以每个用户的行为顺序是登录→交易→（查询）→退出登录。当用户在此过程中发生错误时应该抛出异常，进行合适的处理。在登录结束时应该释放 Java Socket 连接。

9.4.3 场景的设置与运行

在本次性能测试中，用户提出的性能指标不够细致和确切，通过对用户调查和实际业务分析，我们把性能指标的实现方式进行了明确的定位。

1. 并发登录测试场景

并发登录 750 用户/分钟，登录响应时间在 30 秒之内。仔细考虑一下，这里的并发登录750用户/分钟指的是系统能够在1分钟内接受750个用户的登录请求，而处理的效果如何则在交易终端体现，即登录响应时间。基于这样的理解，我们把用户性能指标转化为如下的测试场景：

从第一秒钟开始，用 LoadRunner 每秒钟登录 13 个用户，并保持 Socket 连接，直到 1 分钟结束，从终端向系统一共发送 750 个左右的用户登录请求，系统在一分钟内建立了 750 个连接。在终端观察并统计登录响应时间。如果系统不能响应持续增加的登录请求或平均登录响应时间大于 30 秒，并发登录测试场景都不能算通过。

为了帮助用户更加深入地了解系统的能力，我们对系统的瞬时并发能力进行测试，即测试系统所能承受的最大的瞬时并发用户登录连接请求个数。这个场景通过 LoadRunner 在登录前设置同步点来实现。这个结果将结合上一个结果一同反映系统的登录处理能力。

LoadRunner 场景设置

（1）采用手工场景方案，设置用户数为 750。

（2）对于瞬时并发测试，在 VU 脚本中加入同步点，在 Controller 中设置开启同步点，并选择策略为“100%运行用户到达同步点后再释放”。

（3）对于递增登录测试，在 Controller 的“Edit Schedule”中，选择“Start 13 user every 1 seconds”。

2. 交易流程测试和查询流程测试

在这里我们只对系统的业务负载能力做测试（并发处理能力在登录测试中已经得到考证）。测试场景如下：

在 LoadRunner 中，建立面向目标的测试场景，以 400 笔 Transaction/秒为目标，将调度权交给 LoadRunner 来试图达到这个指标。

LoadRunner 场景设置

（1）“Goal type”选择“Transaction per second”，Goal 是 400 Transaction/s。

（2）“Transaction name”选择“trade”，即交易。

（3）设定最小用户为 50 个，最大用户为 1000 个。

（4）设定如果目标没有达到，即场景退出。

3. 综合测试

交易流程测试和查询流程测试同时进行。

LoadRunner 场景设置

（1）创建两个 Vuser 组，分别执行交易脚本和查询脚本。

（2）在 Controller 的运行时设置中设置“浏览器仿真”，选中“下载非 HTML 资源”和“每次迭代模拟一个新用户”。

4. 健壮性测试（略）



在性能测试开始前，数据库中已经创造出大量的记录，数量级大于每次性能测试场景执行产生的记录。因此，本次性能测试不考虑每次测试之后数据的回滚。

9.4.4 计数器的设置与性能数据收集

在 LoadRunner 的 Controller 中开启 UNIX 系统资源计数器、WebLogic 计数器、DB2 计数器，检测系统资源消耗情况，并最终和测试结果数据合并，成为

分析图表。

9.5　运行场景，得到测试结果和相关数据（Execution 阶段）

9.5.1　脚本的开发

交易业务脚本部分内容如下：

```
Import userpackage; //引入和业务相关的类
public class Actions
 {
        /*登录变量初始化*/
    ProtocolManager protocol;  //ProtocolManager 为实现 Socket 连接的类
    ServiceName service; //ServiceName 对服务端的信息进行了封装，包括 IP 地址和端口号
     LoginMessage login; //LoginMessage 为登录时需要向服务器发送的消息，待服务器确认并返回回应消息时，登录成功
    protocol = new ProtocolManager(); //创建 ProtocolManager 类的 protocol 对象
     service = ServiceName.getInstance();//获得 ServiceName 的实例
     login=new LoginMessage();//创建 LoginMessage 类的 login 对象
    service.setIP("200.31.10.18");//设置服务端的 IP 地址
    service.setPort(17777);//设置服务端的端口号
        /*设置登录消息*/
        login.serUserName(lr.eval.string("{loginName}"));//从数据池里读出用户名，设置在 login 成员变量里

        login.setPasswd("1234");//数据库中添加的用户密码都为 1234
         /*发送登录消息*/
        Lr_start_transaction("login");
        protocol.login(login);//发送登录消息
        lr_end_transaction("login");
        lr_start_transaction("trade");//交易开始点
        TradeMessage trademessage;//生成交易消息
```

```
                /*设置交易消息*/
                ……
                ……
                /*发送交易消息*/
                ……
                ……
                if(sendfail)
                lr_end_transaction("trade", LR_FAIL);//如果发送交易消息失
败，交易结束，返回
                /*循环回收主机返回的处理信息*/
                ……
                ……
                if(recievefail)
                  lr_end_transaction("trade", LR_FAIL);//如果不能接收到主
机处理回应消息，交易结束，返回
                if(recievesuccess)
                  lr_end_transaction("trade", LR_PASS);//如果接收到主机成
功处理的回应消息，交易结束，返回
                ……
    }
```

在上面的例子中，我们主要对每笔交易进行了 Transaction 化。在交易开始时设置开始检测点，交易结束时设置结束检测点，并给 LoadRunner 报出交易状态。在实际的脚本中，在回收交易响应消息时还进行了拆包，在应用层上对交易状态进行识别，并非例子中只在 Socket 层加以判断。

9.5.2 数据的生成

原始环境数据：主要是编写数据库 SQL 脚本，运行插入数据。

测试用例数据：使用 LoadRunner 的 Data Wizard 功能从数据库中导出生成参数列表。

测试结果可在测试执行完毕后，通过 LoadRunner 工具中的 Analysis（分析器）获得。

9.5.3　并发登录测试

依照设计好的测试场景，用 LoadRunner 工具在一分钟内逐渐增加向系统发送登录请求。分别进行三次，结果如表 9-2 所示。

表 9-2　登录测试结果数据表

测试序列	登录成功的用户数	平均响应时间
第一次	485 个用户	25.6 秒/笔
第二次	497 个用户	27.3 秒/笔
第三次	482 个用户	26.2 秒/笔

注：这里的登录成功用户指的是系统接受了登录请求，并建立了连接。平均响应时间在登录脚本里设置检测点，由 LoadRunner 工具自动获得。

考察系统的瞬时并发处理能力：在完成上一步测试的前提下，逐渐增加瞬时并发登录用户数，直到系统极限。

测试执行结果如表 9-3 所示。

表 9-3　瞬时并发登录测试结果数据表

瞬时并发用户数	计划并发数	1	2	10	100	200
	实际并发数	1	2	10	100	122
用户登录响应时间	最短/s	2.0	3.1	7.2	21.2	24.2
	平均/s	2.0	3.3	12.2	24.3	28.1
	最长/s	2.0	3.5	16.5	31.2	33.3

9.5.4　负载测试

1．交易流程测试

测试结果如下：

对于通过网络接口发送的批量业务请求，均在性能指标所指定的时间范围内

得到请求成功的反馈消息，说明主机已经处理成功。

在通过网络接口发送业务请求的同时，开启 IE，通过实际终端界面进行登录和交易，系统响应时间延长，界面显示和刷新明显变慢，到业务量高峰时期，界面已经不能显示任何信息了，处于不可工作的状态。

需要说明的是，系统正常工作时，每个界面终端不仅应该能够展示己方的交易信息，还要展示其他交易单位的交易信息和系统信息。因此，当交易量大的时候，界面需要展示的信息量是巨大的，这本身对终端界面的程序是一个性能考验。

2. 查询流程测试

本流程测试在交易流程测试之后进行，以利用其生成的数据。

测试结果基本满足性能指标。

3. 综合测试

由于交易流程测试未通过，本测试已经不能执行。

9.6 分析测试瓶颈（Adjust 阶段）

9.6.1 并发测试结果分析

根据上述的并发测试响应时间表，我们可以得出以下的结论：

被测系统在一分钟内并不能接受 750 个用户的登录请求，其可接受的登录请求用户数大概为 490 个左右。在这样的条件下，登录响应时间在用户要求范围之内。

被测系统的瞬时并发处理能力约为 122 个用户。

9.6.2　交易流程测试结果分析及性能评价

根据交易流程测试结果可知，通过脚本程序进行业务行为，发送业务请求消息到回收主机处理回应消息，这段时间系统是顺畅的，反应也是迅速的，但是在终端界面却不能即时展现消息。这说明信息的回馈通路在终端界面出现了性能瓶颈。当界面需要在短时间内展示大量交易信息时，已经不能承受负荷。这与开发方采用的胖终端技术有关。

9.6.3　查询流程测试结果分析

查询流程基本符合性能指标。

需要说明的是，实际中，以上每个场景的测试都执行了多次，中间件参数进行了多次的调优。从以上测试的结果分析也可以看出，我们的性能测试瓶颈不是出现在中间件产品上，而是在自身开发的程序上。

10

第 10 章

循序渐进——进阶LoadRunner高手

10.1 性能测试用例的设计策略

10.1.1 “普遍撒网，重点查看”的原则

性能测试是一个复杂的过程，因为它的对象是一个由多个模块甚至多个中间件构成的庞大系统，而每个模块在理论上都有存在性能问题的可能。但我们不可能测遍所有的模块的性能，再给客户出性能报告。事实上，性能测试和其他类型测试一样，遵循软件测试定律：**对软件的完全测试是不可能的**。所以，在做性能测试时我们要做到，设计尽可能少的测试用例来发现尽可能多的性能问题。这样设计出来的用例才是好的用例。

“普遍撒网”的意思就是一个好的性能测试用例的执行应该调用软件系统中尽

可能多的模块。不同的测试用例最好覆盖不同部分的模块。显然，这样做至少有两个好处，一是软件系统被全方位调动起来，容易发现各个环节的问题；二是一次执行，多处测试，节约测试成本。

“重点查看”则要求我们在做性能测试时特别关注软件系统中那些基本的、使用频率高的模块。

“普遍撒网，重点查看”如何实现？下面介绍一种路径分析法。

一提到路径分析法，很多人就会想到白盒测试。而实际上，路径分析法更是一种数学思想，它可应用于解决各个领域的问题，不光是白盒测试。下面，我们将看看路径分析法是怎么应用于性能测试案例设计之中的。

首先是分析软件系统的架构，将客户端发起请求，应用服务器处理，数据入库这个过程以流图的方式勾画出来。可以先从最基本的入手，在最基本流程的基础上再去考虑次要或异常的流程，这样将各种流程逐渐细化，可以逐渐加深对软件系统架构的理解。完成所有流程的图表化后就完成了所有路径的设定。

比如在上章的实例中，就整理出客户端请求有两条主干分支，一个是查询分支，另外一个是交易分支。显然，我们忽略任何一条分支，将会错过发现此分支上节点性能问题的机会。如果我们有足够细致的设计文档的话，完全可以在两个主干分支的基础上，把流图细化到模块级，也就有更多的子分支。我们在这样的基础上设计用例，心里会更踏实和有把握。

找出了所有的路径，下面的工作就是给每条路径设定优先级。我们选取那些优先级高的路径作为我们的性能测试用例；优先级低的路径作为补充案例，可以测，也可以忽略。优先级根据两个原则来选取：一是路径使用的频率，使用越频繁的优先级越高；二是路径的重要程度，如果失败对系统影响越大的优先级越高。将根据这两个原则所分别得到的优先级相加就得到了整个路径的优先级。结合优先级（重点查看）和路径长短（普遍撒网），来选取哪些路径作为我们的性能测试用例。

10.1.2 保证数据的有效性

在做性能测试时，写好了脚本，面对被测系统时，有些茫然，不知做什么，该怎样做才能找到性能瓶颈；在执行性能测试时，看到一堆监控数据，执行日志，不知该如何分析哪里出了问题；在把性能测试报告提交给项目经理和开发人员的时候，不能很理直气壮地说出：嗨，这里存在着一个问题，导致整个系统不能 Work。

在这种情况下，如果你不能让自己的技术经验指导你的行为，那么就确信自己至少还有清晰的逻辑来维护自己的判断。

性能测试是一个复杂的过程，因为它的目标是一个由多个模块甚至多个中间件构成的庞大系统。而我们的目标是要从一堆结果数据中分析出瓶颈，前提是：我们不仅要得到正确的数据，而且还要正确地得到数据。是不是有些头晕啊。那就举个例子吧。

有一个组织为了验证手机辐射究竟对人体有没有害处，抽查了 100 个经常使用手机的人和 100 个没有手机的人，对这 200 个人进行了问卷调查，发现有手机的 100 人经常出现疲倦、失眠，而没有手机的 100 人却没有，因此这个组织推断手机的确对人体有辐射伤害。但实际情况如何呢？其实这两组就不是一个完全相同的样本，有手机的 100 人收入高，其工作生活可能比较紧张，应酬比较多，可能会导致疲惫失眠；没手机的 100 人收入低，生活比较平稳。

当然，这个例子是一个学术研究的问题，没有答案，要通过实验和逻辑分析来寻找答案。类似地，软件的性能测试是一个工程项目，目的是寻找性能瓶颈，如果性能测试者有过丰富的工程经验，他做两三组 Case 就能分析出答案，但不凑巧，如果他接手的是一个新系统，或者他本来对 C 很熟悉，却偏偏要测一个 Web 系统，这种情况下，他就只能硬着头皮面对一个存在大量未知知识的世界了，这时就显示出逻辑分析的能力了。

说了半天，再回到正题上来吧。一般性能测试都会有两个阶段，第一个阶段叫做调优，主要是让那些中间件能够以最好的表现跑得飞快；第二个阶段才是寻

找性能瓶颈，一般盯在应用程序上。

> 听见网友 A 喊：我压力测试第一次响应速度慢，调优之后再施压非常快，看来我的调优成功了！网友 B 沮丧地说：调优之后，响应速度还不如调前，看来调优失败了。我想说的是，当你得出这个结论之前，你一定要想想是否有足够的事实和强有力的逻辑支持你的结论。事实上，A 出现的情况往往是第一次压力还没有 Cache，慢点；第二次，Server 和 Client 都有了 Cache，就快起来了。B 出现的情况是第一次施压时给数据库增加了很多的记录，导致第二次变慢了。出现这些误判的原因是，其实我们第一组场景和第二组场景已经不一样了，环境和条件都发生了变化。这样得出的性能测试数据是没有意义的，也无法用来分析瓶颈。

所以，要想说明问题，就要做两组数据，产生的条件和环境完全一样，只改变系统的一个节点，得到数据，才能明确判断是不是此节点影响了整个系统的性能表现。当然追求完全的一致是过于理想化了，实际上只要不要有大的偏差就可以了。

在实际系统中，性能测试的环境和条件有很多。

（1）数据库中的记录数量是不是差不多，在一个数量级上。

（2）各个中间件的配置是不是一样。

（3）Server 是不是有其他人也在占用资源（性能测试最好保证 Server 是 Clean 的）。

（4）Client 和 Server 的 Cache 要被清除掉或关掉。

（5）最好做完一组数据后，各个 Server 都 Restart 一下，释放掉原先的资源（除非你就想在这个基础上测试）。

（6）有时日志不断地变大也会影响 Server 的性能表现，可以从 I/O 指标上看到。

10.2 LoadRunner 高级功能的使用——Web Click Vuser

10.2.1 Web Click Vuser 的产生背景

60%的用户使用 LoadRunner 做 Web 系统的性能测试。据统计，Web 系统性能测试有 70%的时间花在开发和调试脚本上。在 LoadRunner 8.1 以前，我们测试 Web 系统都是使用 HTTP Vuser 来捕获 Web 系统客户端和 Server 之间的交互，而生成基于 HTTP 协议的脚本。但是 HTTP 脚本同其他 Vuser 脚本一样，基于网络协议，又长又难以理解。另外，HTTP 脚本中的关联也是一个费时的工作，要捕获每个 step 的动态数据，并参数化，而在当今流行的胖客户端的架构下，如 Ajax、Java Applet 等，捕捉动态数据是非常困难的，几乎难以实现。

10.2.2 Web Click Vuser 与传统 Vuser 的差别

基于这样的情况，LoadRunner 从 8.1 版本开始使用 Click and Script 技术（实际上是 QTP 技术，QTP 是 MI 的一个基于 Web UI 的功能测试工具），创造了一种新型的 Vuser，叫做 Web Click Vuser。它与其他传统的基于网络协议的 Vuser 是完全不同的，Web Click Vuser 和 WinRunner、QTP 一样，是基于 Web UI 事件的。因此 Web Click Vuser 脚本更加直观，容易理解。用户对 Web 系统的操作被记录成基于 UI 事件的函数，如点击一个 Button，填写一个 Edit Box，而不是一个个 HTTP 协议上的 get 请求、post 请求。因此，Web Click Vuser 能像浏览器一样，去执行 Java Script 和 Applet。

10.2.3 使用 Web Click Vuser

1. 录制

和选择其他 Vuser 一样，这次在协议对话框中选择 Web Click and Script，如图 10-1 所示。

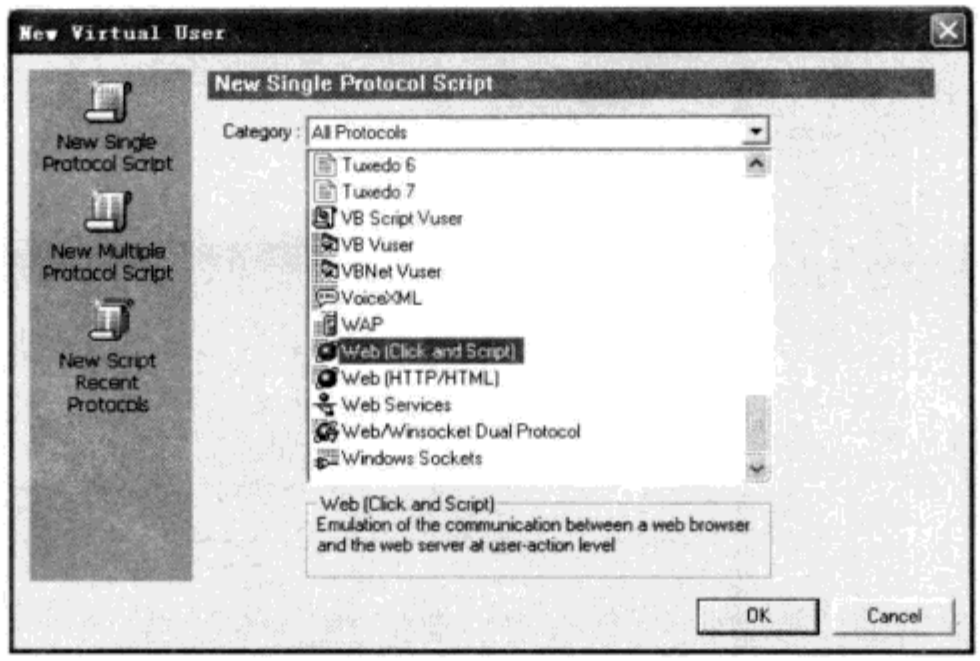


图 10-1　选择 Web Click 协议

录制选项和 HTTP Vuser 一样，如图 10-2 所示。

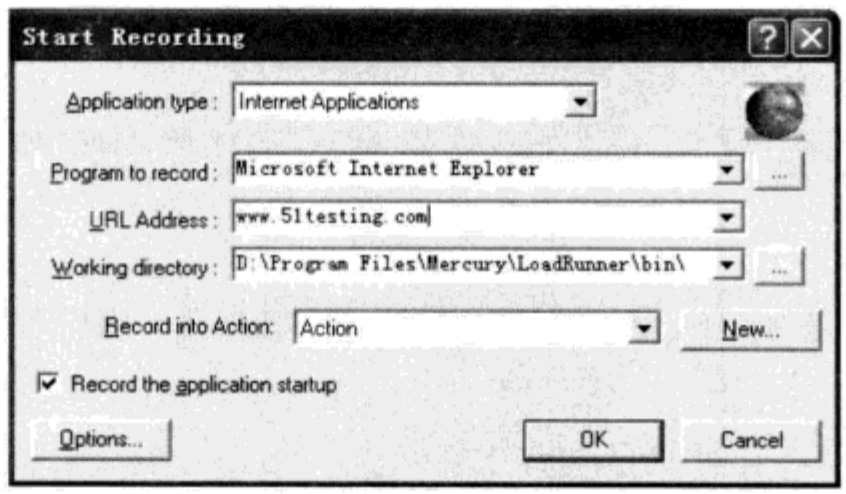


图 10-2　设定 Web Click 的录制选项

我们录制这样一个流程：访问 www.51testing.com 首页→点击“软件测试论坛”→输入用户名、密码→单击“登录”按钮。

在 Tree View 中我们会看到如图 10-3 所示的页面。

在 Tree 中，根下的每个节点代表一个 Web 页面，其下的子节点代表对页面中 Component 的操作。而选中节点的时候，Component 就会被加亮。

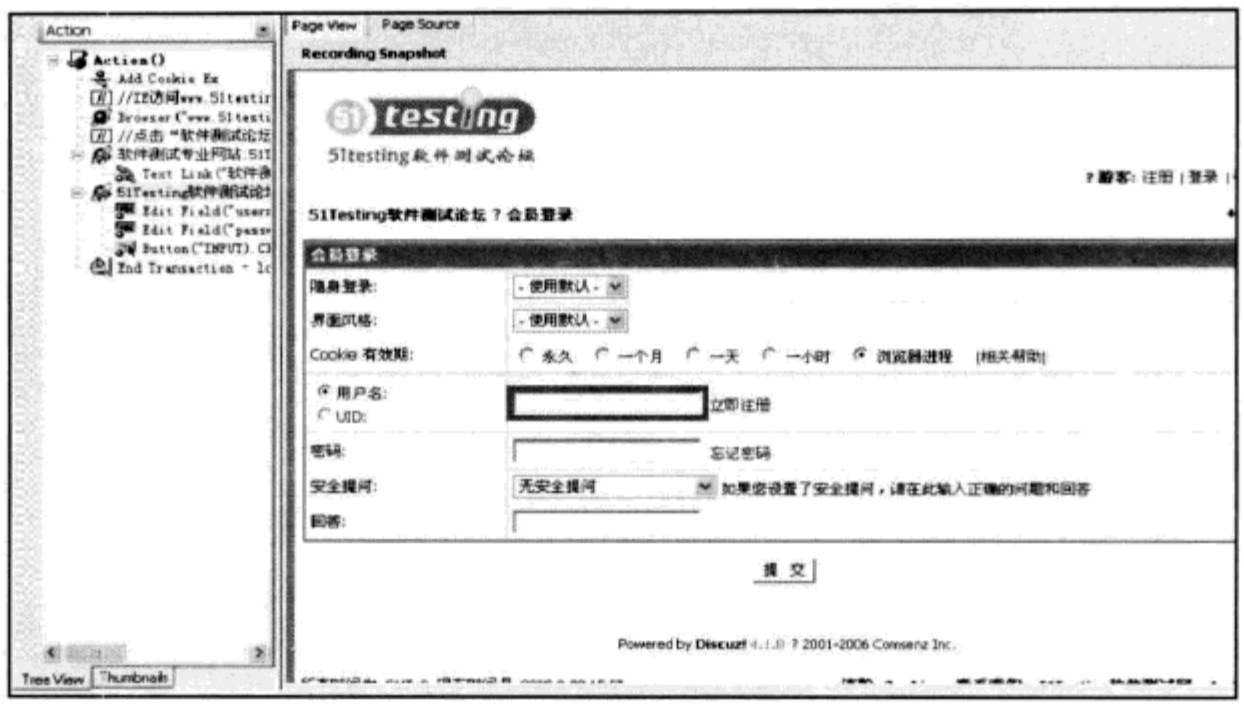


图 10-3 Web Click 脚本的树视图

在 Script View（脚本视图）下，会生成如图 10-4 所示的代码。

```
//IE访问www.51testing.com
    web_browser("www.51testing.com",
        DESCRIPTION,
        ACTION,
        "Navigate=http://www.51testing.com/",
        LAST);

//点击"软件测试论坛"这个链接
    web_text_link("软件测试论坛",
        "Snapshot=t1.inf",
        DESCRIPTION,
        "Text=软件测试论坛",
        LAST);
    web_edit_field("username",
        "Snapshot=t3.inf",
        DESCRIPTION,
        "Type=text",
        "Name=username",
        "BrowserOrdinal=2",
        ACTION,
        "SetValue=sunshinelius",
        LAST);
    web_edit_field("password",
        "Snapshot=t4.inf",
        DESCRIPTION,
        "Type=password",
        "Name=password",
        "BrowserOrdinal=2",
        ACTION,
        "SetEncryptedValue=44f29d41df9348232321c445f3c7b6",
        LAST);
    web_button("INPUT",
        "Snapshot=t7.inf",
        DESCRIPTION,
        "Type=submit",
        "Tag=INPUT",
        "ID=",
```

图 10-4 Web Click 的脚本视图

我们可以看到，使用 Web Click Vuser 录制生成基于 C 的脚本，在脚本函数中，有一些新“面孔”：

- web_edit_field：在一个 Eidt Box 中填写数据；
- web_text_link：点击一个链接；
- web_button：点击一个 Button。

同时，Web Click Vuser 还是会记录网络协议级别上的事件，用户可以随时从 Web Click Vuser 脚本转换到 HTTP Vuser 脚本。

在菜单“Tools”>“Regenerate Scripts”>“Options”>“Recording”中设置，如图 10-5 所示。

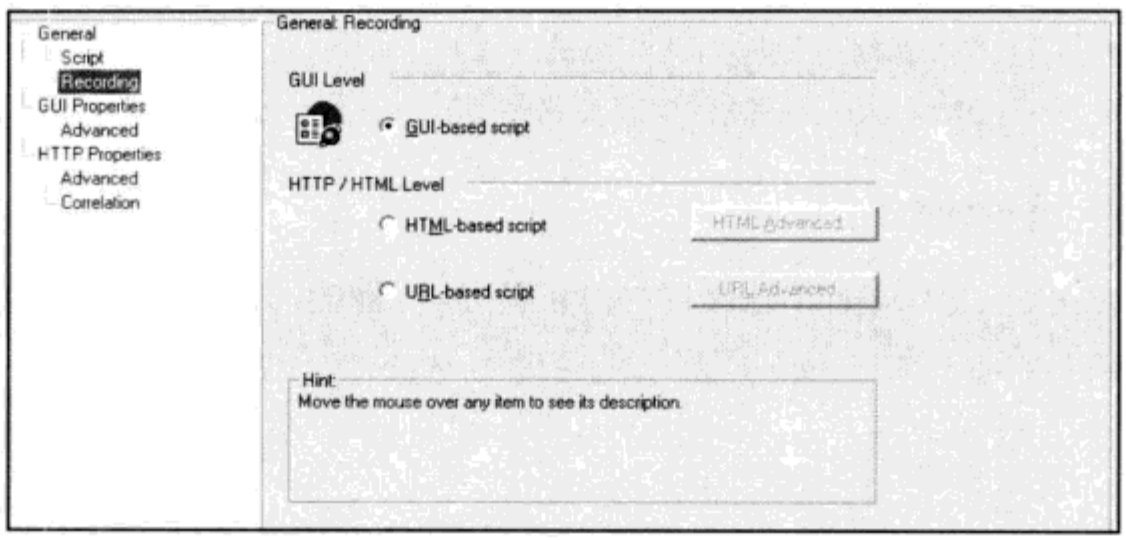


图 10-5　设定脚本转换

2. 回放脚本

Web Click Vuser 可以像我们操作浏览器一样执行脚本，非常方便。

下面介绍一下 Web Click Vuser 的用途、限制以及 License。

（1）用途

目前，Web Click Vuser 的运行环境是 LoadRunner 8.1＋SP2。在 HP 网站中的 patch list 中叫做 FP2。

可以看出 Web Click Vuser 是基于界面对象的脚本，那么它的好处显而易见，那就是它度量的是一个完整的系统时间，从客户端界面发起请求一直到服务器的响应返回。有网友使用过 Web Click Vuser 后，发现同样事务的响应时间比 HTTP

Vuser 慢了 0.2 秒左右，恰恰就是这个原因。

（2）限制

由于 Web Click Vuser 是 LoadRunner 新推出的功能，还不是很稳定，也有不少需要完善的地方，如下：

- 只支持 IE 录制（Windows）。
- 一个 Load Generator 只可运行数量不高的 Web Click Vuser（目前 200 个 Vuser 是测试通过的）。
- 目前 Web Click Vuser 只支持英文。
- 对 Applet、ActiveX 控件的支持不够。

（3）License

Web Click Vuser 被要求和 HTTP Vuser 同样的 License，也就是说，如果你有了 HTTP Vuser 的 License，那么就可使用 Web Click Vuser 了。

11

第11章

定量分析及诊断——建立性能度量模型

以 LoadRunner 为主导的性能测试可以获知被测系统的基本性能信息，比如：

- 被测系统在指定环境下，是否达到预期性能指标？
- 被测系统是否存在性能瓶颈？

这只是对被测系统的一个定性判断，而作为性能测试的专业人员，我们需要给客户更详细、更有建设意义的信息，比如：

- 性能瓶颈存在于被测系统的哪个节点、模块甚至代码行？可以给出解决建议吗？
- 当前系统经过性能调整后，可以减少多少响应时间？
- 被测系统上线后，随着数据容量的增加和硬件环境的变化，是否可能遇到性能拐点，又如何应对？
- 如何在当前的开发测试过程中，避免此类的性能问题将来再次发生？

等等。这些问题实际上促使性能测试进入了一个定量分析的层次。

简单来说，定性分析主要是“发现问题”，而定量分析不但要“定位问题”，最好还能“解决问题”，甚至要在当下“避免问题”，在将来“预测问题”。

对软件系统做性能定量分析是一个高级软件人员的优秀素质，这不仅仅是在LoadRunner 层次上纯熟地使用技巧，更需要软件各个领域的深厚专业知识，而且还要有和各个团队角色耐心细致交流的能力。

11.1 实现性能度量的准备工作

11.1.1 性能度量

显而易见，能够实现性能定量分析的前提是要有数据，而且是详细而全面的性能数据。就像一个医术高明的大夫，往往会在诊断前和病人谈话交流，在充分了解病人的体质、症状、病史之后，才能对症下药，因人施方。

度量数据有哪些呢？

1. 拓扑节点度量数据

针对一个 B/S 四层架构的系统，比较典型的有：

- 网络性能数据

包括当前局域网网络带宽，测试过程中网络的平均流量、峰值流量、占用带宽百分比等。

- Web 服务器 HTTP 请求性能数据

包括 HTTP 服务器的基本配置、线程池数目、http Get/Post 请求的个数、响应时间等。

- 应用服务器交易性能数据

交易定义因具体应用而定，一般包括处理的 transaction 的个数、transaction 的处理最大时间、平均时间等。

- 数据库监控报表

数据库的基本配置信息、top SQL、BreakDown 等。

……

由上可见，节点度量数据一般都会在系统各个节点上进行采集。这是因为，度量分析的根本目的是将定性测试中得到的整个系统响应时间进行细分，需要知道到底哪个环节模块消耗的资源最大，占用的响应时间最长。**我们能分析并定位瓶颈到什么层次，取决于度量数据采集点下钻到达的位置。**

2. 基准环境度量数据

在节点度量数据的基础上，将同样的性能测试场景运行在不同的软硬件基准环境下，得出系统的基准环境度量数据，它会反映当前软件系统在何种配置环境下获得最优的性能表现。

比如，在不同数据容量配置下，对在线文件管理系统进行性能测试（如表 11-1 所示）。

表 11-1　数据容量性能度量

性能场景	200K 基础数据	400K 基础数据	600K 基础数据	800K 基础数据
Upload File	223 ms	355 ms	458 ms	552 ms
Download File	84	115	151	199
Search File	54	61	62	66
Delete File	74	77	87	90

3. 周期迭加度量数据

在节点度量数据的基础上，将同样的性能测试场景运行在软件产品生命周期中各个可测版本上，得出被测产品的周期迭加度量数据，它会反映当前软件系统

性能随着版本的更新而变化的趋势。

比如，在不同测试版本上，对在线文件管理系统进行性能测试（如表 11-2 所示）。

表 11-2 版本递增性能度量

性能场景	8 月 3 日 版本	8 月 31 日 版本	9 月 10 日版本	9 月 24 日版本
Upload File	223 ms	253ms	254ms	278ms
Download File	84	97	90	97
Search File	74	76	78	80
Delete File	54	54	55	58

11.1.2 度量方式

有了度量数据后，我们将采用不同的方式对其进行分析，来达到性能度量的目的。

1. 使用下钻细分法进行瓶颈定位

我们用层层下钻的方式来进行性能的定量分析

一级下钻

某交易的系统响应时间=客户端处理时间+网络时间+Web 服务器时间
+应用处理时间+数据库时间



案例分析

例子：比如某邮件系统的 Web 发送邮件的总共耗费时间为 4 秒，根据度量数据，进行一级下钻：

- 客户端处理时间：浏览器处理时间，忽略
- 网络响应时间：54ms，相比 15s，可以忽略
- Web 服务器时间：0.56s/Http Request
- SMTP 服务邮件发送处理时间：未知
- 数据库处理时间：connect time+Sql parse time+ sql execute time=1.4s

总响应时间=页面时间+网络时间+Web 处理时间
+SMTP 邮件服务处理时间+数据库处理时间
4s=0+0+0.56+ SMTP 应用服务时间+1.4
SMTP 应用服务时间=4–0.56–1.4≈2s

二级下钻

下面尝试将 SMTP 服务时间进一步下钻细分。

SMTP 服务架构如图 11-1 所示。

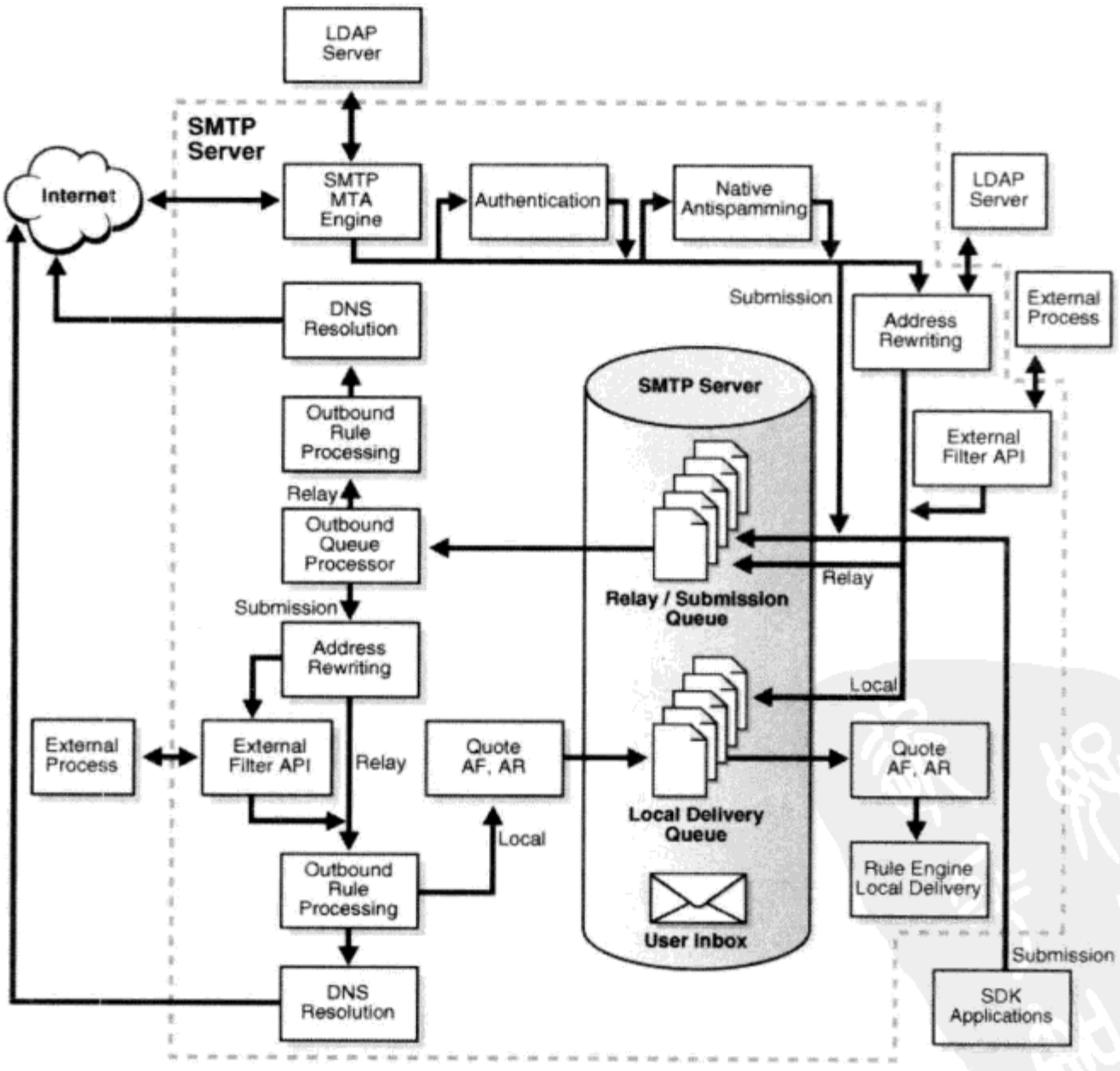


图 11-1　SMTP Service Architecture

大致可把 SMTP 处理分为几个阶段：

SMTP 服务时间=认证时间+SMTP Local Queue 入队时间+Outbound 处理时间

在各个模块上进行监测，时间细分为 2s≈0.02s+1.2s+0.7s。

由于 Job Queue 上花费了 1.2s，那么说明邮件队列还有调优的空间。

2. 使用正交图来预测性能拐点

两个因素 A 和 B，B 的变化会导致 A 的变化，那么可以以 A 为纵轴，B 为横轴，建立二维正交图来观察 A 和 B 之间的影响作用大小及发展趋势。

比如在上面的基准环境度量数据例子中，A 为性能响应时间，B 为基础容量数据，以 upload file 测试案例为例，可以绘制如图 11-2 所示的曲线图。

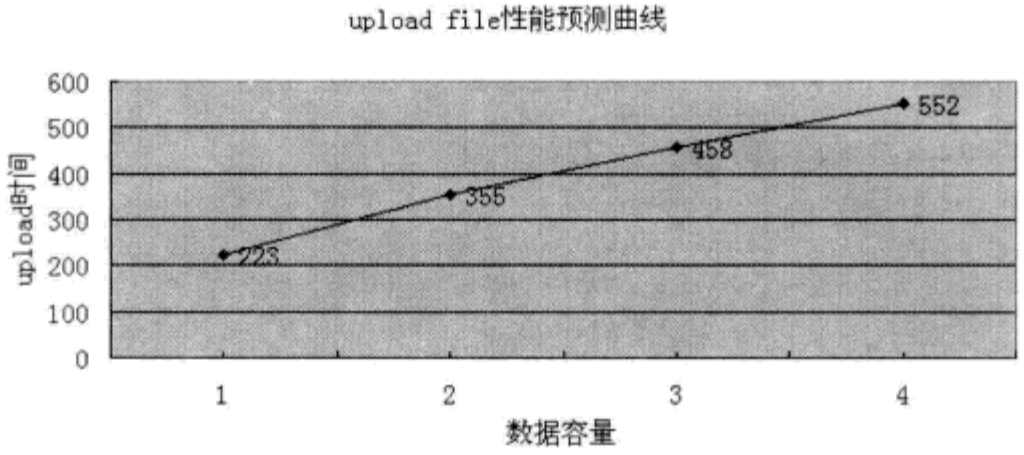


图 11-2 容量—性能变化曲线

从以上曲线可知，随着数据容量的增加，性能响应时间增加幅度较少，这是一个斜率小于 1 的曲线，因此，可以预测在将来容量增加的一段时间内，应该不会存在容量拐点和瓶颈。

在上面的周期迭加度量数据例子中，A 为性能时间，而 B 为版本周期。以 download file 测试案例为例，可以绘制如图 11-3 所示的曲线图。

上面的曲线反映了一个问题，第二个版本相比第一个版本出现了较大幅度的性能衰退，而在第三个版本和第四个版本中得到解决，并加以控制。

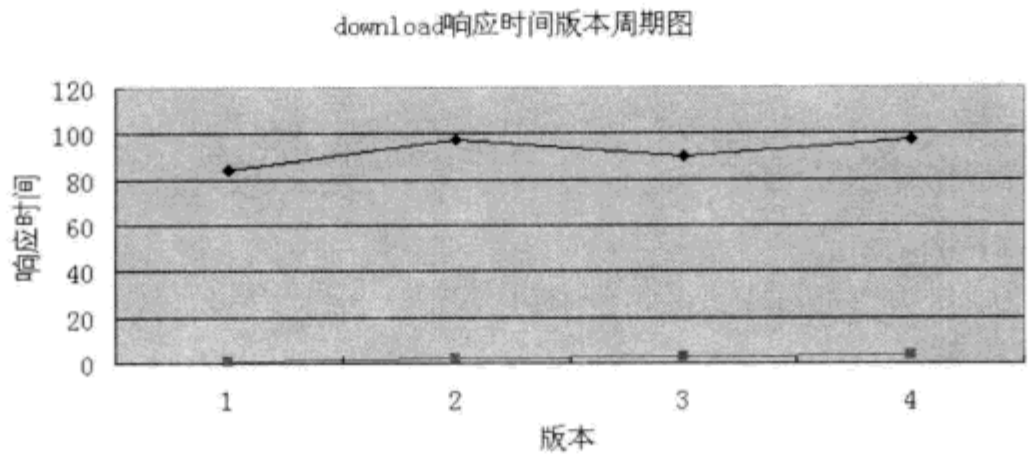


图 11-3　版本—性能变化曲线

总结

数据是性能量化分析最重要的基础和前提。要想做到量化分析，需要在不同的时间和空间内重复执行性能测试，并收集详细而全面的度量数据。**从某种程度上说，没有采集度量数据的性能测试是毫无意义的。**

案例分析　性能 bug 定量分析实际案例

Bug 标题： 1.1 版本升级到 1.2 后 login action 发生 40%的性能衰减。

测试环境： 某 Web 订票系统 1.2 版本。

场景描述： 50 用户 ramp 并发 login 运行一个小时，thinktime 500ms。

测试结果： 1.2 版本 Login Service 花费时间为 4.2s，而 1.1 版本同样的场景案例则花费 2.4s。

对 1.2 版本上采集的性能度量数据进行下钻细分，Oracle DB 的 AWR TOP SQL report 捕捉到一条 SQL 语句耗时达 1.5s。

```
select username,password from ocs_data.auth_users where
NLSSORT(username,'NLS_SORT=GENERIC_M_CI') = NLSSORT('ocsadmin',
'NLS_SORT=GENERIC_M_CI')
```

对比 1.1 版本的 AWR Report，发现这是一条 1.2 版本新引入的 SQL 语句。

1.1 版本同样作用的 SQL 语句为：

```
    select   username,password   from   ocs_data.auth_users   where
username= 'ocsadmin'
```

耗时仅为 50ms。

解决建议：

新 SQL 语句耗时较长，是由于 where 使用了新的字段表达式，而使得原有的字段索引没有排上用场。建议建立新的字段索引。

Bug 总结：

上面的 bug 分析逻辑清楚，直接揭示瓶颈根源，定位到了语句级别。能完成这样的量化分析，依赖于以下几个基础条件：

- 节点数据采集点详细而全面，在 oracle DB 上使用 awr 进行 SQL 语句采集，这为 SQL 语句性能分析提供了数据基础。
- 周期数据持续采集形成对比基准。在 1.1 和 1.2 版本中持续进行性能回归测试，直接将新旧版本上 top SQL 进行对比，这为瓶颈定位提供了逻辑条件。

11.2 案例实践——性能测试第一阶段 Ajax 页面基准性能分析

11.2.1 页面基准分析目标

基准分析是 Ajax 性能测试项目实施的第一阶段，其主要目的是考察单个用户使用系统时，用户获得的性能体验，包括响应时间和资源的使用情况，并通过相关工具对性能表现进行细分（breakDown），从而提前发现客户端和应用服务器上可能存在的性能问题甚至瓶颈，为第二阶段的并发负载测试提供对比参考依据和技术基础。

11.2.2　分析所使用的工具

本阶段主要采用 HttpWatch 和 IEWebDeveloper 两种 Web 客户端细分监控工具来完成基准分析。

1．HttpWatch

HttpWatch 是一款 HTTP 网络协议监控分析工具，可以分解各个 http request 的具体响应情况，包括 Send，Receive，Method，Status Code，URL 等，如图 11-4 所示。

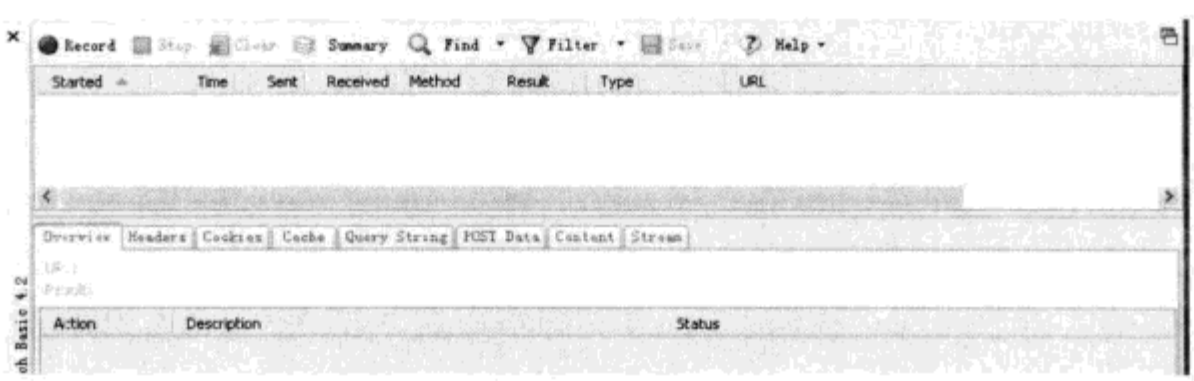


图 11-4　HttpWatch

2．IEWeb Developer

IEWebDeveloper 是一款性能 Web 页面 DOM 分析工具，能够分解页面的各个元素，包括 html 的标准元素和 JavaScript 脚本等，如图 11-5 所示。

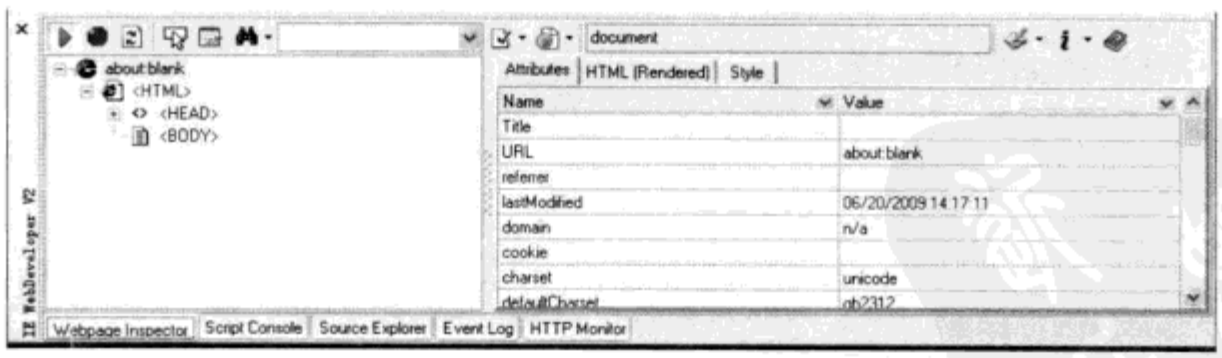


图 11-5　IEWebDeveloper

11.2.3　术语揭示

- **Download Data**：从 Server 向客户端的下行数据。

- **Upload Data**：从客户端向 Server 的上行数据。
- **Total Time**：完成一次页面切换或提交所花费的时间。
- **JavaScript**：运行在浏览器中的脚本语言。
- **BreakDown Time**：细分时间。

11.2.4 基准测试案例设计及运行

1. 项目介绍

某电力信息管理系统负责对所在区域的供电、配电生产等系统上千个节点进行管理验收等工作。本系统使用 Extjs 框架开发 Web 页面，后台使用 WebLogic 应用服务器和 Oracle 数据库服务器。

Extjs 框架是比较流行的一种 Ajax 应用模式，代码表现为一套开源的跨浏览器的 JavaScript 脚本库，依据 Extjs 可以构建多种 Rich Web 客户端应用。

但从系统性能的角度来说，使用 Ajax 模式会引发一个性能考虑，那就是由于浏览器中嵌入了大量的 JavaScript 逻辑处理运算，使得客户端也有成为性能瓶颈的潜在风险。因此在大并发负载测试开始之前，有必要对客户端的 JavaScript 进行性能基准分析。

本轮测试对将要负载测试的功能点进行基准分析：

（1）登录信息系统。

（2）点击“电票管理”，展开票单信息页面。

（3）点击“新增”，提交票单。

注：以上测试均在系统没有负载的情况下，以单用户完成。

2. Ajax 页面登录功能基准分析

本环节的诊断对象为登录功能。即从登录页面输入用户名、密码到登录成功，展现系统主页面结束，监控客户端与服务器之间的交互。登录后页面如

图 11-6 所示。

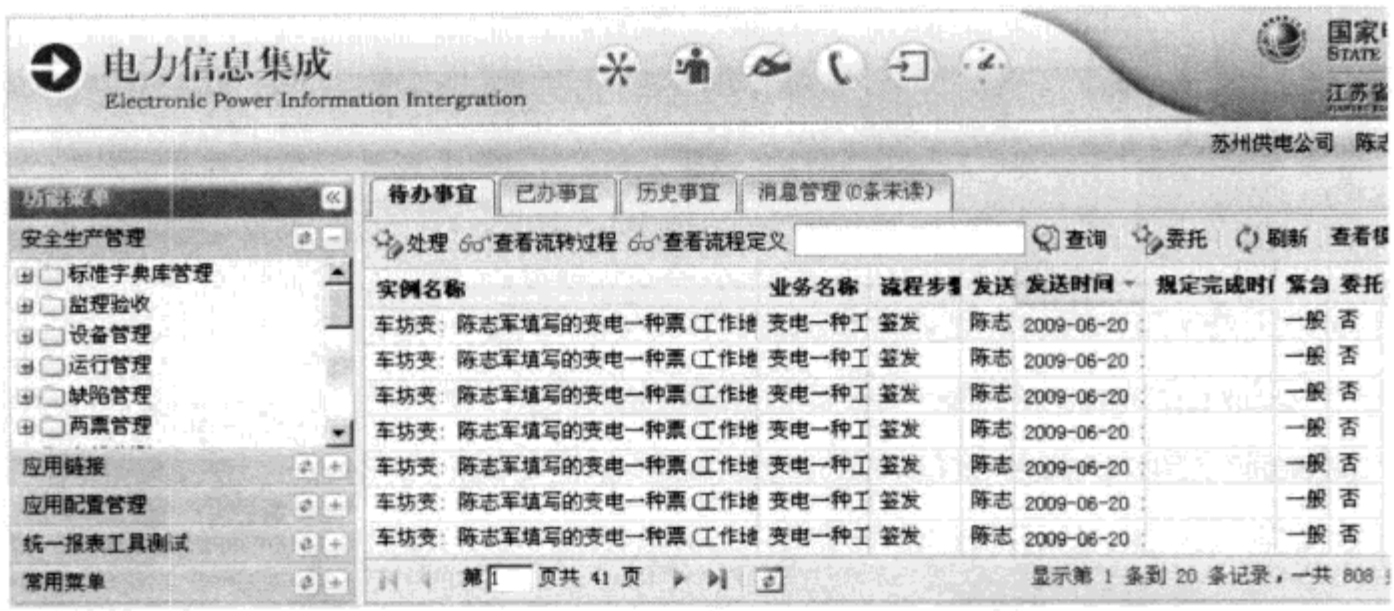


图 11-6　登录

在此过程中，使用 HttpWatch 和 IEWebDeveloper 采集到的性能数据如表 11-3 所示。

表 11-3　性能数据

指标	性能表现
单用户登录花费时间	8.123s（秒）
发生的 http request 个数	188 个
网络下行数据流量	1.9MB
网络上行数据流量	114KB
占用网络带宽平均值	2MB/8.123s = 246.214KB/s
占用网络带宽峰值	462KB/s

对 8.123s 的登录时间进行细分诊断，结果如表 11-4 所示。

表 11-4　细分诊断

诊断对象	总共	细分过程	注
登录响应时间	8.123 s	发生七次对 Server Action 的调用 总共花费时间 2.223s	其中捕捉到 main.do 的 getFuncNode 被重复调用 5 次
		下载 83 个 java scirpt 文件 总共花费时间 3.413s	其中捕捉到 ext-all-debug.js 文件高达 1.05MB

续表

诊断对象	总共	细分过程	注
登录响应时间	8.123 s	下载 JPG 图片等其他资源 总共花费时间 1.45s	

注 1：被重复调用 server 的 action 全路径为

http://192.168.1.1:8080/appframe-web/main.do?method=getFuncNode

注 2：1.05MB 的 js 文件全路径为

http://192.168.1.1:8080/appframe-web/ext2.2.1/ext-all-debug.js

可见，单用户登录系统所花费的时间高达 8s，占用网络带宽高达 462KB/s，多消耗在下载 js 文件上。

3. 票单管理功能

本环节的诊断对象为点击“票单管理”。即在左侧树形导航栏中点击“票单管理”，客户端与服务器之间发生的交互，页面如图 11-7 所示。



图 11-7 票单管理

在此过程中，使用 HttpWatch 和 IEWebDeveloper 采集到的性能数据如表 11-5 所示。

表 11-5　性能数据

指标	性能表现
点击“变电一种票”所花费时间	4.017 s（秒）
发生的 http request 个数	45 个
网络下行数据流量	2.5 MB
网络上行数据流量	23 KB
占用网络带宽平均值	2.5MB/4.017 s =622.354 KB/s
占用网络带宽峰值	823.423 KB/s

对 4.017s 的登录时间进行细分诊断，结果如表 11-6 所示。

表 11-6　细分诊断

诊断对象	总共	细分过程	注
点击“变电一种票”所花费时间	4.017 s（秒）	发生三次对 Server Action 的调用 总共花费时间 0.269 s	时间属于正常
		下载 45 个 java scirpt 文件 总共花费时间 3.178 s	其中再次捕捉到 ext-all-debug.js 文件被下载，高达 1.05MB
		下载 JPG 图片等其他资源 总共花费时间 0.824 s	

注：在前期登录步骤中已经下载过的 js 文件，在本次操作步骤中再次从 server 上下载。1.05MB 的 js 文件全路径为 http://192.168.0.1:8080/appframe-web/ext2.2.1/ext-all-debug.js

分析：大尺寸的 js 文件被重复下载。

4. 提交票单

本环节的诊断对象在提交票单时，到票单提交成功后，客户端与服务器之间发生的交互页面如图 11-8 所示。

在此过程中，使用 HttpWatch 和 IEWebDeveloper 采集到的性能数据如表 11-7 所示。

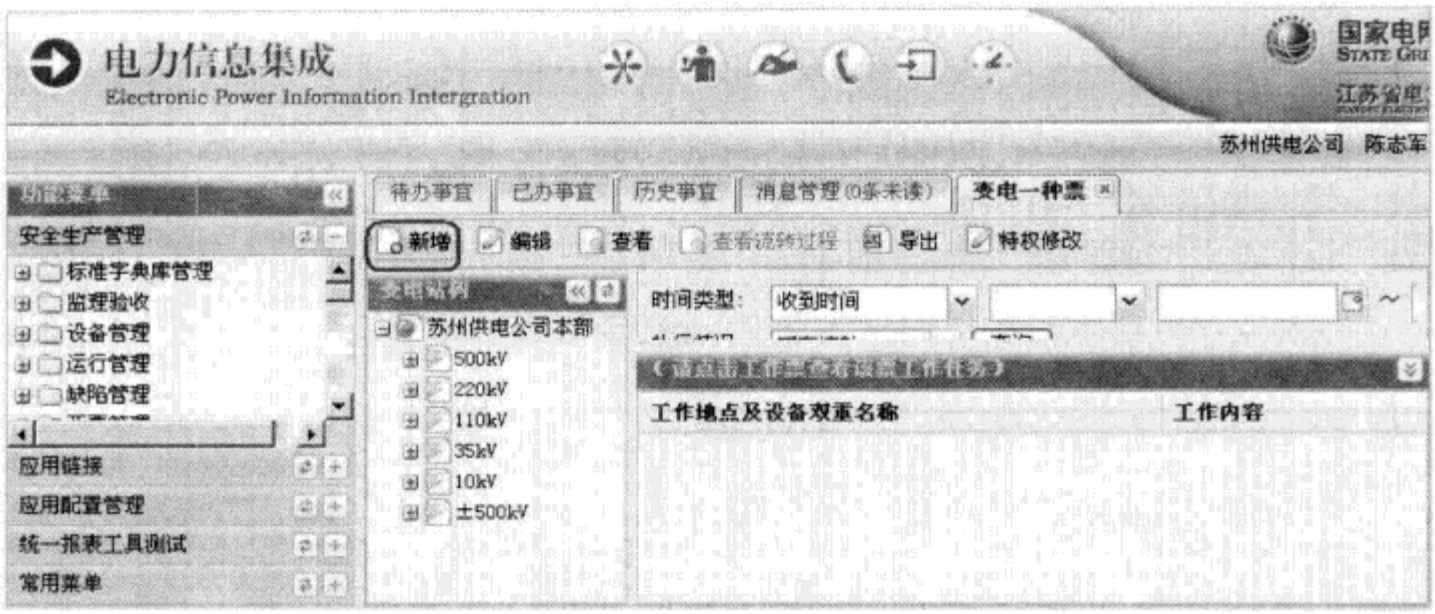


图 11-8 提交界面

表 11-7 性能数据

指标	性能表现
点击“新增”所花费时间	14.265 s（秒）
发生的 http request 个数	5 个
网络下行数据流量	7 KB
网络上行数据流量	3 KB
占用网络带宽平均值	10KB/14.265 s =701.016 B/s
占用网络带宽峰值	1.2 KB/s

对 4.017s 的登录时间进行细分诊断，结果如表 11-8 所示。

表 11-8 细分诊断

诊断对象	总共	细分过程	注
点击“新增”所花费时间	14.265 s（秒）	发生一次对 Server Action 的调用 总共花费时间 0.072 s	时间属于正常
		下载 JPG 图片等其他资源 总共花费时间 0.012 s	时间属于正常

分析：本次操作时间长达 14.265s，但实际上与服务器之间的交互时间却很小（不到 1s）。因此可以推断，长达 10 多秒的时间极有可能花费在客户端 JavaScript 文件执行上。

11.2.5　结果分析

1．潜在性能瓶颈

通过响应时间分析，可以得知目前系统中存在严重的客户端和网络瓶颈。主要表现为：

- JavaScript 文件过大，其中 ext-all-debug.js 文件高达 1.05MB，严重占用网络带宽。
- JavaScript 文件过多，消耗了大量的 http request 资源，导致总体响应时间延长。
- JavaScript 文件重复下载，大量的 JavaScript 文件在与 Server 交互的过程中被重复下载。

以上三种现象综合作用，消耗了用户的时间和网络资源的一半之多，已经构成了性能瓶颈。

2．网络瓶颈预测

在百兆局域网内，完整带宽为 100M b/s，转换成 Byte，为 100/8=12.5 MB/s。

根据经验，一般网络带宽瓶颈参数为 0.7，即占用网络带宽 70%以上，即可视为出现网络瓶颈。

因此，实际有效带宽为 12.5MB/s * 70% = 8.75MB/s。

在上面基准分析中，“新增”环节占用带宽峰值为 823.423 KB/s。

因此在百兆局域网内，可容纳的并发用户数为 8.75M/823K= 10.6 个用户。

推论：目前系统在百兆局域网环境下，用户数超过 10.6 个，即可造成网络瓶颈，网络瓶颈的表现为系统响应速度变慢，但实际上应用服务器却是空闲状态。

11.2.6 优化建议

（1）精简 JavaScript 代码，尤其是 ext-all-debug.js 文件。

经查看 ext-all.debug.js 文件源码，发现大量引用开源代码，其中有一部分属于无效代码，比如对 Safari 浏览器的支持和处理（Safari 是 Mac 操作系统的浏览器）。

（2）合并 JavaScript 文件

从步骤一和步骤二的性能表现对比，可以看到：

登录步骤，下载 js 文件 83 个，总共 1.9MB，耗费时间 3.413 s。

因此，在 js 文件总体大小无法压缩的情况下，可以通过合并和减少 js 文件个数进一步提升性能。

（3）采用本地和 Server 缓存解决重复消耗的时间

js 文件采用本地的 temp 目录 cache 方法，避免相同的大尺寸文件重复下载。

同样地，对于频繁调用的 server action，也应该在 Server 上通过开辟 cache 内存的方法来提升 Server 的性能。

（4）对于步骤三出现的问题，客户端执行 js 文件消耗 10s 是极不正常的，建议开发人员重新检查并优化相关代码。

11.2.7 预测

通过以上数据分析可以预测，如果有效解决 js 文件问题，至少能够提升 30% 的系统性能。

由于网络瓶颈的影响，在本轮性能问题没有解决的情况下，不宜开展并发负载测试的执行。

11.3　案例实践：性能测试第二阶段 Ajax 系统负载测试

11.3.1　本轮测试目的

经过第一阶段基准分析后，已经初步发现和定位存在于 Web 客户端和网络的性能缺陷。因此本阶段着重从整个系统的角度来验证软件系统的性能需求是否能够达到既定指标。

11.3.2　测试场景规划

本阶段测试采用 LoadRunner 手工设置场景，负载模式设为 Ramp up，即批量递增加载模式。使用 Schedule Builder 设置如图 11-9 所示。

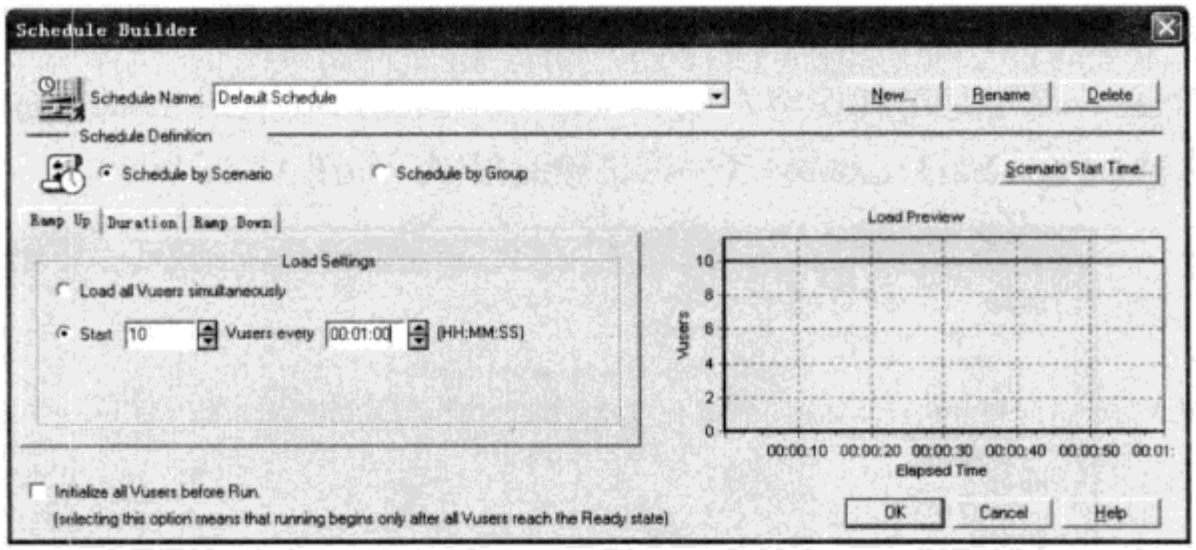


图 11-9　Schedule Builder 设置

Ramp up 模式为每隔 2 分钟递增加载 10 个用户，预计 20 分钟完成 100 个用户的加载。Duration 设置如图 11-10 所示。

当加载目标完成后，场景将会持续运行 10 分钟，然后退出。

Ramp up 加载模式优势：压力均匀充分，最大程度模拟用户使用真实场景。从 10 个用户到 100 个用户并发，能够完整有效地采集数据，准确找出性能曲线趋势。能够兼顾瞬时并发和负载两种模式，即批量加载点体现瞬时并发模式，加载过程则体现为负载模式。

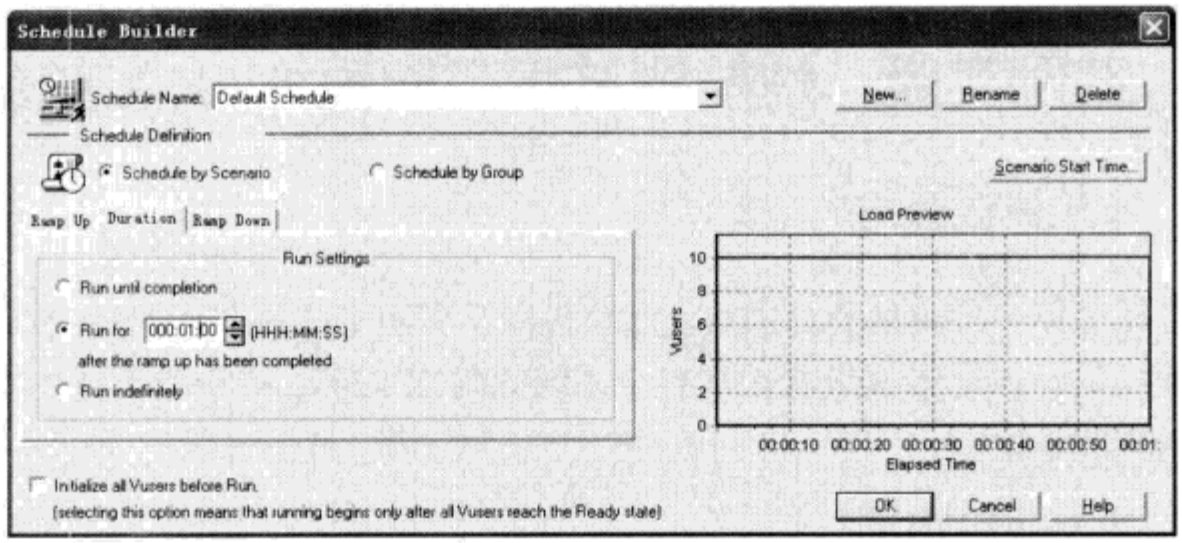


图 11-10 Duration 设置

在经过第一阶段基准分析后，已知客户端和网络存在着严重瓶颈，导致在100MB 局域网内实际上只能并发 10.6 个用户。在开发人员没有解决此问题的情况下，我们必须保证在本阶段并发负载压力能够充分传递到 Web 服务器和应用服务器上去，而不是消耗在客户端和网络上。因此，我们在 Loadrunner 中采用了 download non-html resource 开关来消除此限制，如图 11-11 所示。

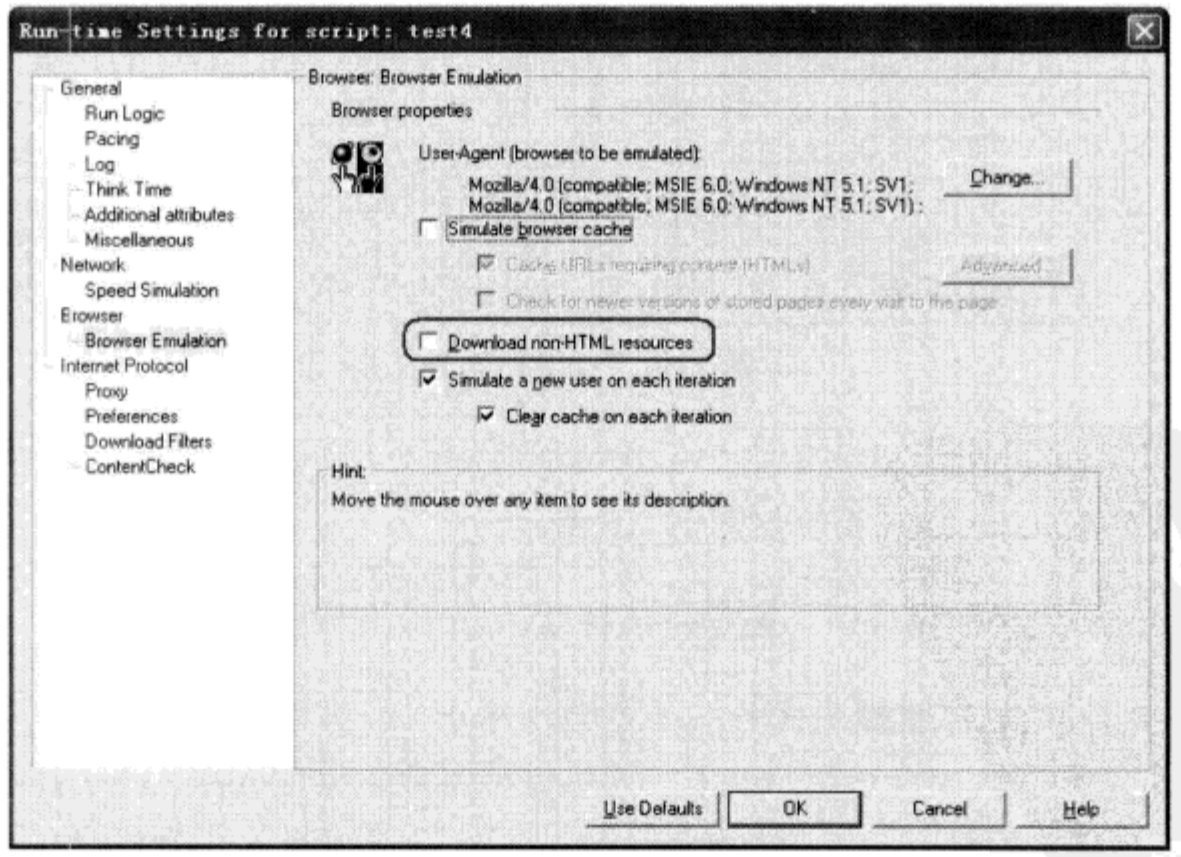


图 11-11 设置

作此设置后，LoadRunner 在运行负载测试时，将不会从服务器下载 JavaScript 文件和相关图片，从而消除可能的网络瓶颈，确保压力施加到服务器侧。

11.3.3　负载测试案例分析

1. 某票单功能测试结果概要（表 11-9）

表 11-9　功能测试结果概要

性能指标项	性能指标值	
并发虚拟用户数	100 个	
网络总吞吐量	750 496 535B	
平均占用带宽	3 014 042B/s	
总共 HTTP 请求数	36 489 个	
总共运行时间	4'8"	
功能分解交易	新增	成功 100 个，平均响应时间 49.473s
	刷新	成功 100 个，平均响应时间 1.878s
	查看详细	成功 100 个，平均响应时间 24.005s
	提交	成功 100 个，平均响应时间为 63.595s

在以上运行中，无失败或错误事务，但需要注意的是，100 个并发用户在近 4 分钟内已经造成了 750MB 的流量，占用带宽 3.1M/s。

在以上运行中，无失败或错误事务，但需要注意的是，Loadrunner 显示已经顺利并发 100 个用户，并成功进行提交。但经过查阅后台数据，发现仅成功提交 31 个票单。经过反复排查，发现在 100 用户并发时，数据库出现死锁，报出信息如下：

> 信息：执行查询 SQL 语句时发生错误:[IBM][CLI Driver][DB2/AIX64] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code "68". SQLSTATE=40001
>
> [IBM][CLI Driver][DB2/AIX64] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code "68". SQLSTATE=40001

2. 虚拟用户数 ramp 趋势图（图 11-12）

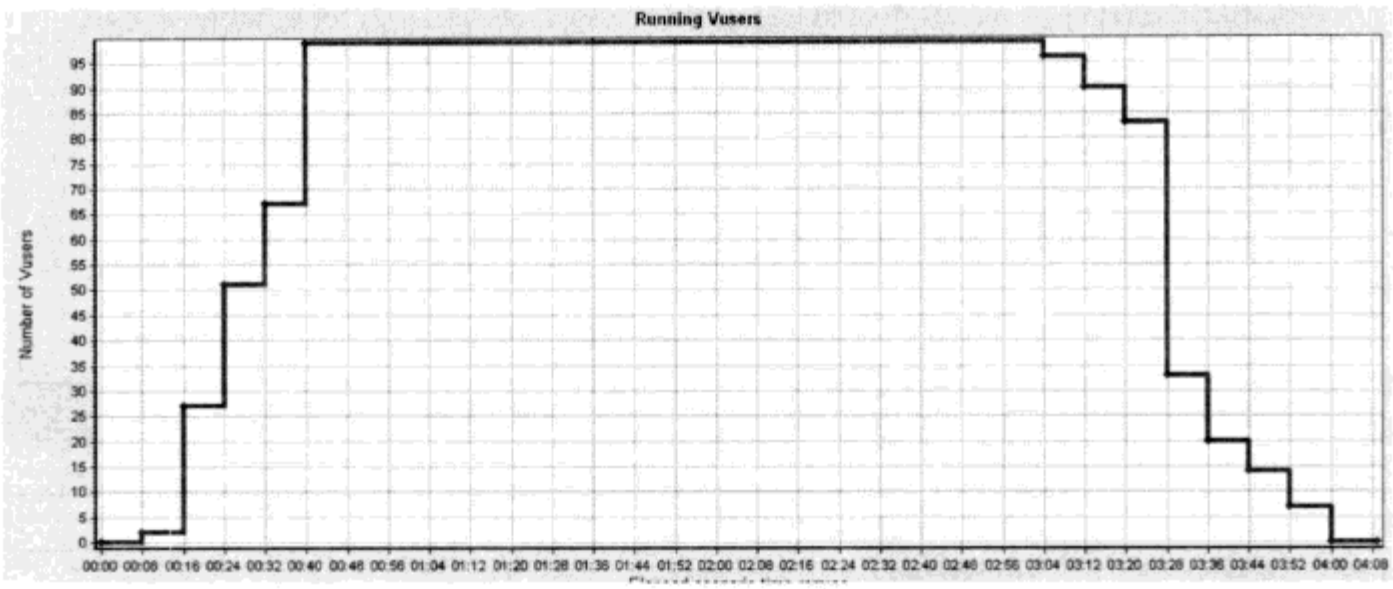


图 11-12 虚拟用户数 ramp 趋势图

3. DB2 Server UNIX Resource 趋势图（图 11-13）

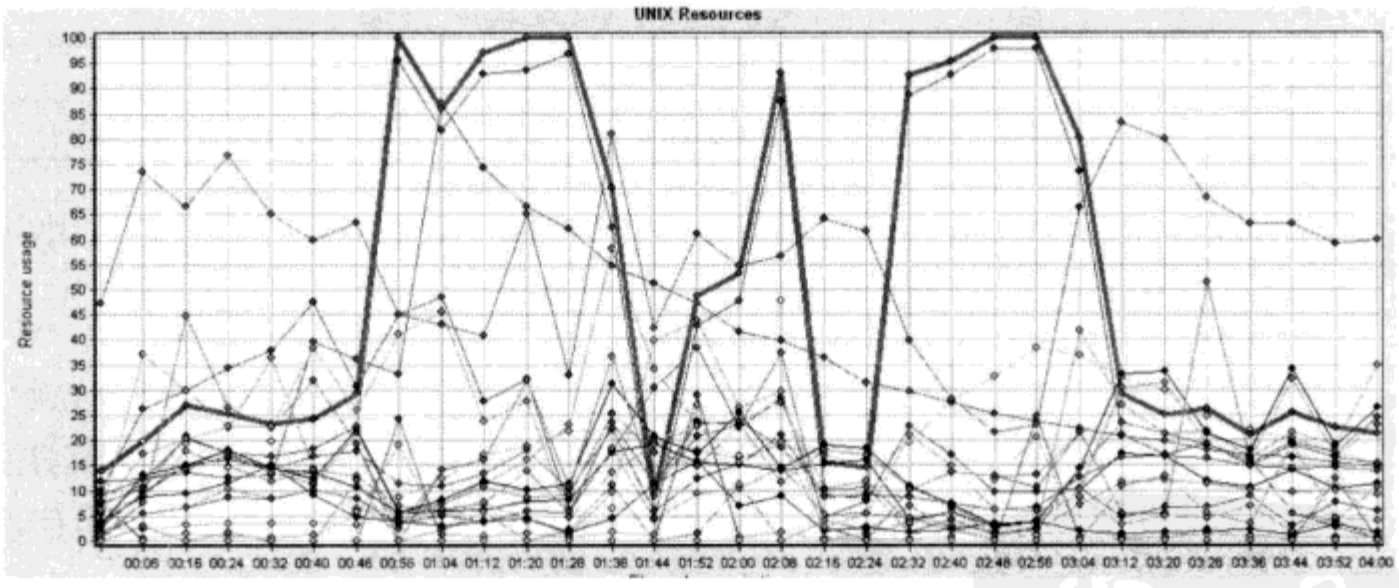


图 11-13 DB2 Server UNIX Resource 趋势图

图中浑色高位曲线为 CPU Utilization（UNIX Kernel Statistics），与并发用户数趋势基本一致。

4. WebLogic 监控图

WebLogic 的 ExecuteThreadTotalCount 一直处于 100 的占用率，此处有调优

空间，如图 11-14 所示。

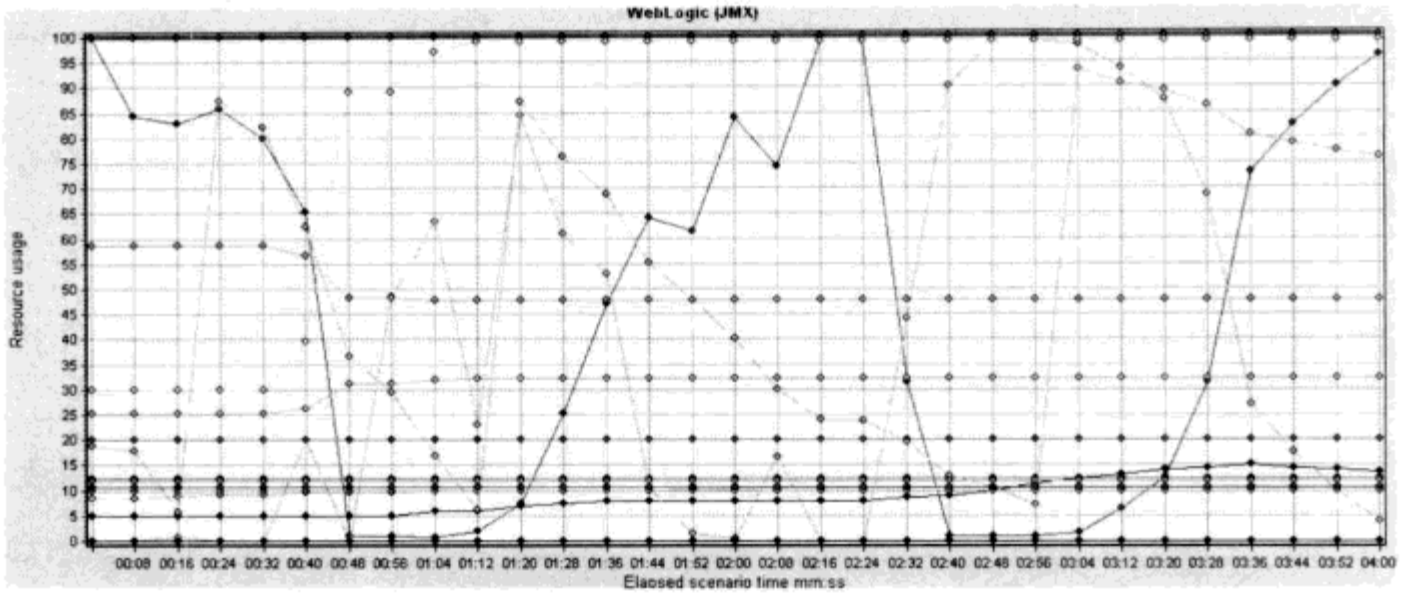


图 11-14　WebLogic 监控图

5. JVM 监控图

从图 11-15 可见，JVM 的 heap size 随着压力的起伏而收放，可以判定本系统应该没有明显的内存泄漏问题。

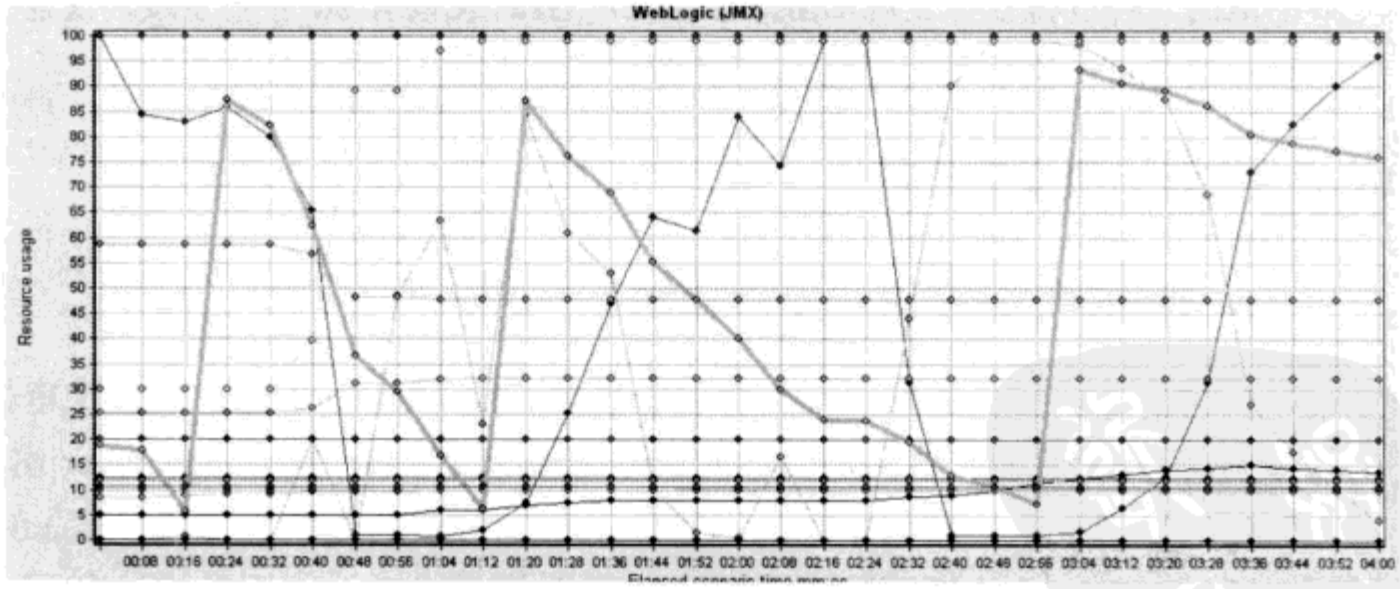


图 11-15　JVM 监控图

6. DB2 监控图

图 11-16 中高位曲线为 total_sort_time 是排序时间，为 986584，Total_sorts 为

33078。

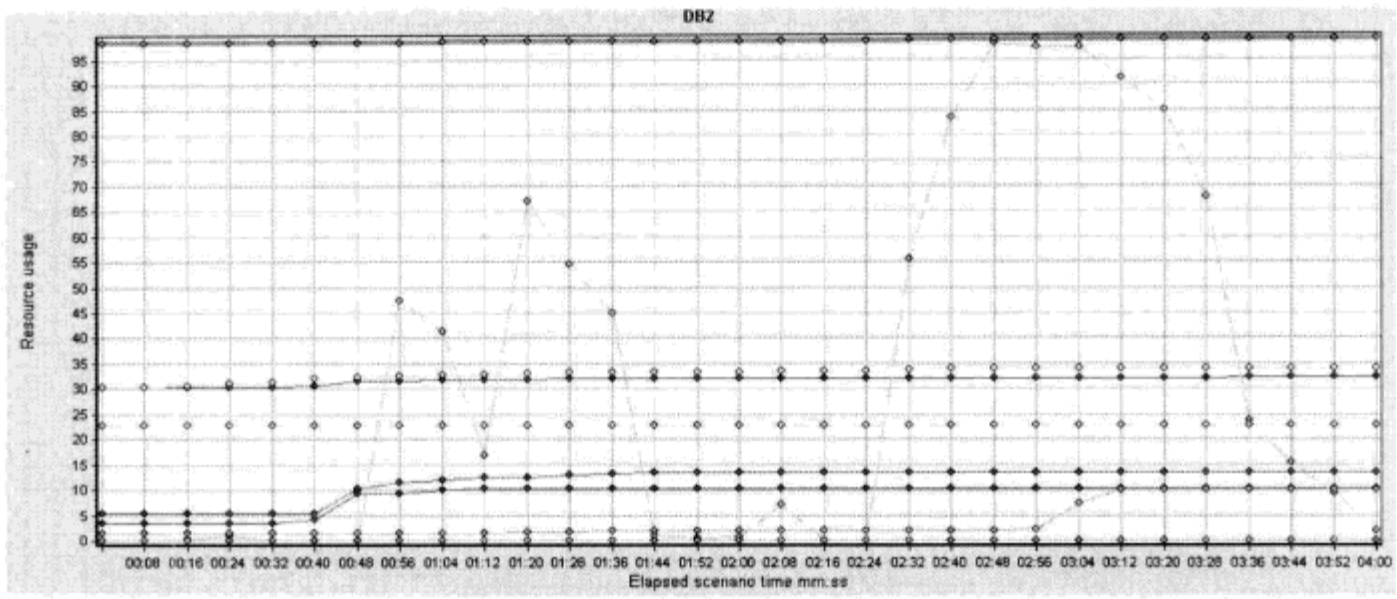


图 11-16 DB2 监控图

排序时间为 total_sort_time/total_sorts=986584/33078=30，花在排序上的时间过长。疑为不规范 SQL 语句所致，有待排查。

图中最低位曲线为 pool_index_read，是索引池的读取率，一直处于零，这是不正常的。经过排查，发现 USERID.T_LP_GZP 表包含 80 多个字段，仅建了 7 个索引，很有可能索引建得不合理，没有建在 order by 字段上。

11.3.4 性能结果量化分析

1. 数据库死锁问题及相关解决建议

当执行 100 用户并发审批两种票场景时，LoadRunner 提示 100 个审批事务全部成功通过，而通过应用查询，实际提交数目却为 60 个。LoadRunner 出现“误判”是由于 HTTP 协议的本身的限制和 HTTP 协议的无状态特点，LoadRunner 只能通过 HTTP 协议返回码来判断脚本是否执行成功，而不是在业务层次上做判断。

经排查，发现在并发负载下，提交审批两种票时，失败的页面上出现了如下错误信息：

> 信息：执行查询 SQL 语句时发生错误:[IBM][CLI Driver][DB2/AIX64] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code "68". SQLSTATE=40001
>
> [IBM][CLI Driver][DB2/AIX64] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code "68". SQLSTATE=40001

从 DB2 的监控数据来看，在并发数目超过 30 后，lock time 大大增加，超过 10s，最终导致死锁超时。

问题的可能根源在：

（1）开发人员在提交 SQL 语句，做 update，insert 等操作时，启用了排他锁，导致死锁。

（2）数据库的 locktimeout 参数设置过小。

在针对 locktimeout 参数做过优化调试后，成功提交数有所提升，但仍无法从根本上消除死锁问题的发生。因此，建议在应用程序 jdbc 层上修改代码，避免死锁发生，才是解决问题根本之道。

具体解决建议：

建议开发人员检查 jdbc 层上可能存在死锁语句，这有可能：

- 一次 commit 包含多条 SQL 语句
- 涉及同时多表操作的 SQL 语句
- 逻辑死锁

2. 数据库表索引设计相关问题及解决建议

数据库在设计上存在如下两个问题：

（1）表设计存在大表风险。

在测试过程中，发现工单和两票在做删除操作时，记录只是改变状态字段，而始终分别保存在 USERID.T_GDMX_GD，USERID.T_LP_GZP，USERID.T_LP_CZP 表中。

这可能会导致在生产环境下，随着运行时间和用户数的增加，如上表数据记录越来越多，最终成为大表，引起查询性能的下降。一般，业界通常的解决类似问题的办法是另建备份表保存被删除的记录。

（2）索引构建不合理，没有起到相应的作用。

在运行两票性能测试场景过程中，通过 loadrunner 的 db2 performance counter 监控捕获 **pool_index_read** 指标，发现此指标的值一直为 0。因此，我们推断当前数据库中存在着索引构建不合理的问题。

已经查证，两票的工单存放于一张数据库表中，此表有 80 多个字段，而索引仅为 7 个。

另外，通过变电一种票的性能测试结果可以看到 total_sort_time 总共花费 986 秒，排序时间过长，说明数据库的索引没有建在 order by 的字段上。

具体解决建议：

与开发人员进行沟通，构建合理的表及索引。

3. 硬件可能瓶颈

在负载测试过程中，曾发生过数据库服务器 CPU 100%运行，memory 居高不下，最终数据库服务宕掉。经检测，在操作系统中有过虚拟内存分配高达 9GB 现象，导致机子负荷过重。

具体解决建议：

鉴于此，我们建议实际生产环境上的数据库服务器内存至少 16GB 或更高。

11.4　总结

从以上案例可以看出，对性能的定量分析可以在性能测试的不同阶段对不同的系统节点进行。而产品性能测试相比项目性能测试要更加复杂，产品性能测试还要加入回归周期、版本验证等因素。因此，产品的性能测试同功能测试一样，应该遵循以下几个原则：

（1）尽早开始，在整个产品测试计划之初，同时开展性能测试规划，并让性能测试团队提早参与到整个产品的需求评审、设计评审、代码走查等活动中去。

（2）每一轮性能回归测试，需要充分记录环境信息，采集更多的度量数据，以供回归测试进行参考比对分析。

（3）形成良好的可重复的性能测试过程，有助于产品性能质量的改进。

12

第12章

测试也开发——自开发性能测试框架

Loadrunner 作为一个强大的性能测试工具，几乎可以渗透到各种应用软件领域中。但在一些特殊应用背景下，用户还是需要考虑自开发性能测试工具，比如：

（1）软件系统运行需要特殊应用环境，非 Loadrunner 能支持，如嵌入式操作系统。

（2）特殊应用协议，而开发语言非 Loadrunner 兼容，比如 C++开发的电信网关协议。

（3）商业成本考虑，Loadrunner 价格高昂，而企业产品又有相对充足的时间，可以规划开发自己的性能测试工具。例如，某企业在开发一个周期长达七八年的通信领域应用平台，并发用户数经常过万，Loadrunner 并不能胜任，完全可以考虑自开发性能测试工具。

在这种情况下，自开发性能测试框架就非常有必要了。下面我们就谈谈性能测试工具开发的一些设计思路和实现方向。

12.1　性能测试框架架构设计

12.1.1　技术考虑的准备

目前业界有不少自动化测试开发人员，心头灵光一现出一个想法，就立刻开始编码，经过几天雕琢加工，也完全有可能开发出一段像模像样能运行的测试程序。一般这样产生的测试程序应急用还是可以的，如果要想在自动化测试中长期稳定地运行、扩展并维护，往往难以胜任。

其实，开发测试工具和开发软件没有本质的区别，也应该遵循需求分析、架构设计和编码等软件过程。

1．开发语言的选择

若计划开发一个稳定的性能测试工具，应该首先考虑到开发语言的可重用性，对网络协议的支持程度、可扩展性、运行效率、跨平台运行等需求。

Java 就是一种很好的面向对象语言，并且有相应的开源 package，可支持多种网络协议，如表 12-1 所示。

表 12-1　Java 支持的网络协议

网络协议	Java 相应组件库	提供接口
Web HTTP 协议	org.apache.commons.httpclient.HttpClient	HTTP 协议相应的 put，get，post 等操作原语
邮件服务 SMTP/IMAP 协议	javax.mail.Message	有关邮件 send，receive 等操作
FTP/FTPS 网络协议文件传输	org.apache.commons.net.ftp.FTPClient	FTP 文件上传等操作
NNTP 网络新闻传输协议	org.apache.commons.net.nntp	NNTP 协议操作原语
POP3 邮件接收协议	org.apache.commons.net.pop3	Pop3 协议操作原语
Telnet 远程访问协议	org.apache.commons.net.telnet	telnet 协议操作原语

续表

网络协议	Java 相应组件库	提供接口
TFTP（Trivial File Transfer Protocol,简单文件传输协议）	org.apache.commons.net.tftp	Tftp 协议操作原语
Finger（用户信息协议）	org.apache.commons.net.finger	Finger 协议操作原语
Whois（域名查找协议）	org.apache.commons.net.whois	Whois 协议操作原语

这样 Java package 组件就为网络协议实现提供了技术保障。而 Java 的另外一个跨平台特性则可以保证程序能够在不同的操作系统上运行，比如 Linux，Windows，UNIX 和 Solaris 等多个主流操作系统平台。

Java 的跨平台是通过 Java 虚拟机（JVM）来实现的。Java 应用程序的开发周期包括编译、下载、解释和执行几个部分。Java 编译程序将 Java 源程序翻译为 JVM 可执行代码——字节码。这一编译过程同 C/C++的编译有些不同。当 C 编译器编译生成一个对象的代码时，该代码是为在某一特定硬件平台运行而产生的。因此，在编译过程中，编译程序通过查表将所有对符号的引用转换为特定的内存偏移量，以保证程序运行。Java 编译器却不能将对变量和方法的引用编译为数值引用，也不确定程序执行过程中的内存布局，而是将这些符号引用信息保留在字节码中，由解释器在运行过程中创建内存布局，然后再通过查表来确定一个方法所在的地址。这样就有效地保证了 Java 的可移植性和安全性。

简言之，只要在不同操作系统上安装了相应版本的 JVM（虚拟机），那么 Java 应用程序代码完全可以无缝移植。

对于我们的性能测试工具来说，好处是显而易见的，Java 跨平台可以使得压力从任何一个平台主机上发起，不管是 Linux，Windows 还是 Solaris 等。

2. 多线程同步技术

性能测试必然会要求多线程同时发起网络请求。这里对于性能测试工具开发人员说，面临着几个十分关键的技术问题。

- 通过何种方法实现线程之间的同步

- 如何完成线程之间的通信和交互
- 结果数据的统计与分析

通常，在技术层面上，两种线程通信方法最常见。

方法一：通过访问共享变量的方式。

方法二：通过管道流。

这两种方法各有利弊，有一定的适用领域。在本章中，我们将引入另外一种线程通信的方法，即通过对象继承技术建立通信通道。

12.1.2　Automation Center 性能测试框架设计实例

下面我们将以 Automation Center 为例，介绍性能测试框架设计思路和实现原理。

1. UML Class 设计图（图 12-1）

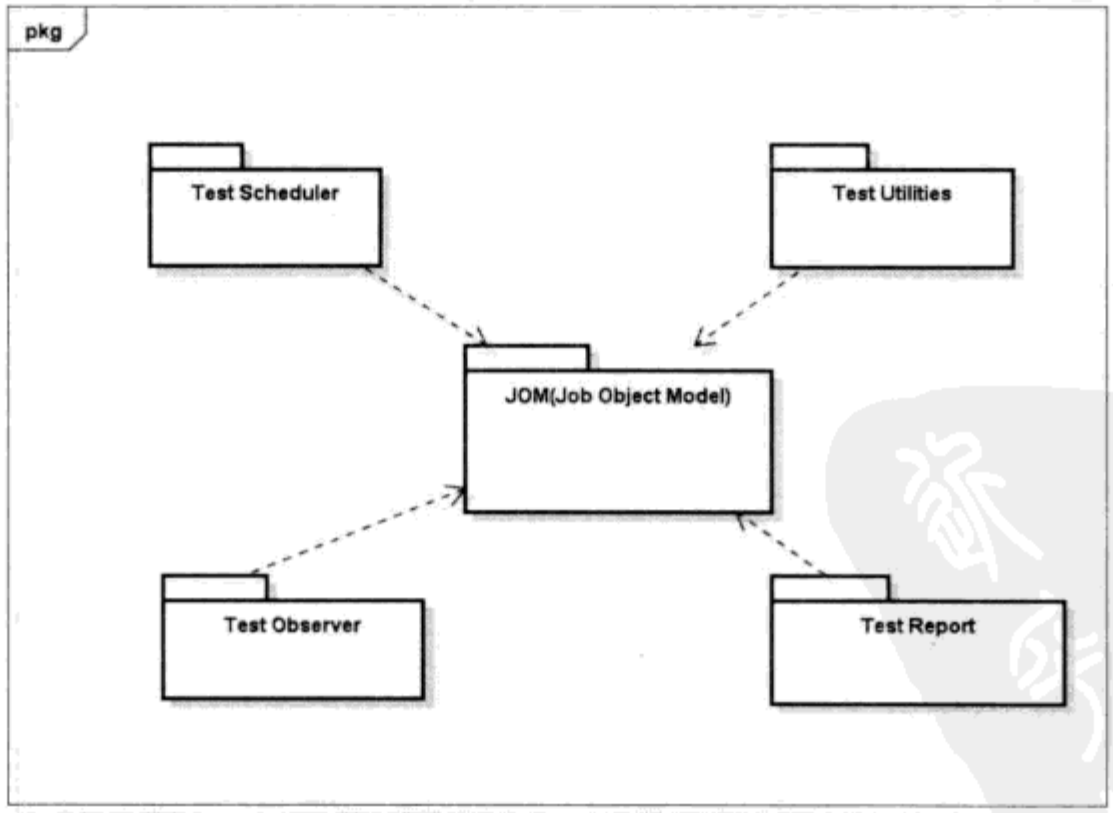


图 12-1　Automation Center Class 设计图

在 AC 性能测试框架的架构设计里，以测试任务（Job Object）为核心。分别有 Test Scheduler（性能测试规划），TestUtilities（功能库），TestObserver（监控）和 TestReport（性能测试报告）作为支撑。

2. Job Object Model（性能测试任务对象模型）

TestJobObject Model 是 AC 的运行核心，它是一个树形结构，每个 TestJob 下包含多个属性和对象，结构如图 12-2 所示。

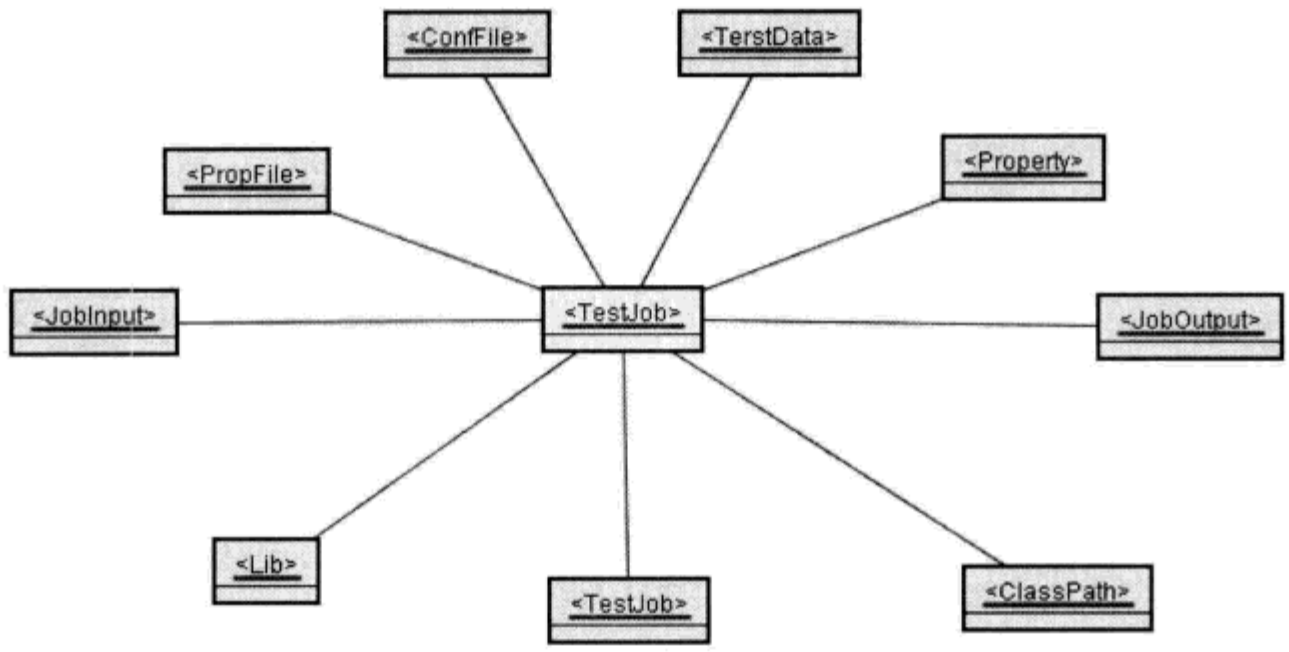


图 12-2 TestJobObject 架构图

在并发测试中，一个 TestJob 就是一个用户的单笔事务。

TestJob 将会做如下属性定义：

- **Name**：事务的名称。
- **Description**：事务的描述信息。
- **Depends**：事务的依赖信息。比如在事务 A 的 depends 定义为事务 B，则 AC 会首先运行事务 B，并根据 B 的成败，判断是否执行 A。
- **Driver_type**: 此为事务的引擎类型，AC 会根据此字段启动相应的 JobEngine，类似于 loadrunner 的 vuser type，比如 http engine，smtp engine 或其他自定义类型的事务。

- **StartTime**：记录事务的开始时间，以毫秒为单位。
- **EndTime**：记录事务的结束时间，以毫秒为单位。

一个 TestJob 还可以包含以下一个或多个附属对象：

- **<JobInput>**：测试任务输入信息，在本 Test Job 开始执行之时，AC 会检查其所有 JobInput 是否已经准备好。
- **<JobOutput>**：测试任务输出信息，在本 Test Job 结束执行之后，AC 会检查其所有的 JobOutput 是否都已经按照定义输出。
- **<PropFile>**：环境文件，遵循 Java Properties 定义规范，和测试环境相关的信息，比如被测服务器地址、测试账号及口令等。
- **<ConfFile>**：AC 配置文件，遵循 Java Proeprties 定义规范，和任务设置相关的信息，比如测试超时时间、重复运行次数等。
- **<TestData>**：测试数据，支持 XSL 和 XML 两种数据格式。AC 会在 Test Job 执行的时候批量加载性能测试数据，供 Test Case 使用。
- **<ClassPath>**：Java Test Job 专用，AC 在运行 java job 时，会调用 classloader 将所引用到的 jar 包装载至 JVM 中。

另外，Test Job 同时还是一个递归的自定义结构，这意味着一个 Test Job 下可以递归包含若干个子 Test Job，这种 Test Job 又称做抽象 Job，AC 在运行抽象 Job 时，会首先递归运行其下的所有子 Test Job，然后分析这些 Test Job 的结果状态，最终确定抽象 Job 的运行结果状态。

抽象 Job 和实体 Job 在定义时唯一不同之处是，实体 Job 会明确指定测试引擎类型。而抽象 Job 则没有指定引擎类型。AC 会根据不同的 Job 类型启动相应的测试引擎。

TestJob 的自递归定义，使得 AC 可以灵活地适应在软件系统任何层次上定义事务。比如，可以将一个 SMTP 发邮件的操作分解到命令原语层次上，并进行事务定义和度量，Helo，Auth Login，Mail From，Data 都可以被定义成一个 Test Job（事务）进行分析。而各个 Test Job 可以通过父 Job 进行通信和同步。

3. Test Scheduler（性能测试规划器）

Test Scheduler 负责设置性能测试场景，并将 Test Job 按照既定策略调用运行，如图 12-3 所示。

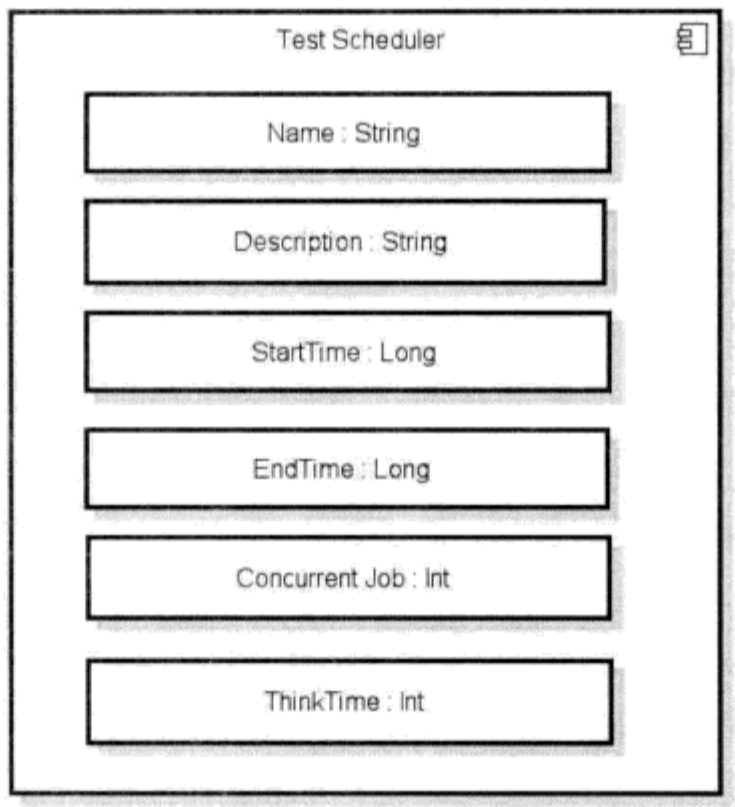


图 12-3 Test Scheulder 属性图

TestScheduler 可以做如下定义。

- **Schedule Name**：测试计划名称
- **Schedule Description**：测试计划描述
- **StartTime**：测试运行开始时间
- **EndTime**：测试运行结束时间
- **ConcurrentJob**：并发测试事务数量（并发线程数）
- **ThinkTime**：思考时间

Test Sceduler 采用 Java 的对象克隆技术来解决线程通信问题，当 TestJob 被指定并发 100 虚拟用户运行，则 Test Scheduler 实际上会将 TestJob 对象克隆 100 次，并调用 TestJob 各自的 run 方法。这样，每个 TestJob 运行各自独立，耦合度降到

最低，而同时由于克隆出来的 Test Job 隶属于同一父 Test Job，因此可以通过父对象进行数据交互和通信，这样设计上大大地减少了常规多线程并发所可能带来的冲突风险，而且使得线程通信渠道更加畅通。

4．Test Utilities（性能测试功能库）

Test Utilites 负责为 Test Job 提供各种通用功能 API，使得自动化测试开发人员更加容易上手开发测试脚本，如图 12-4 所示。

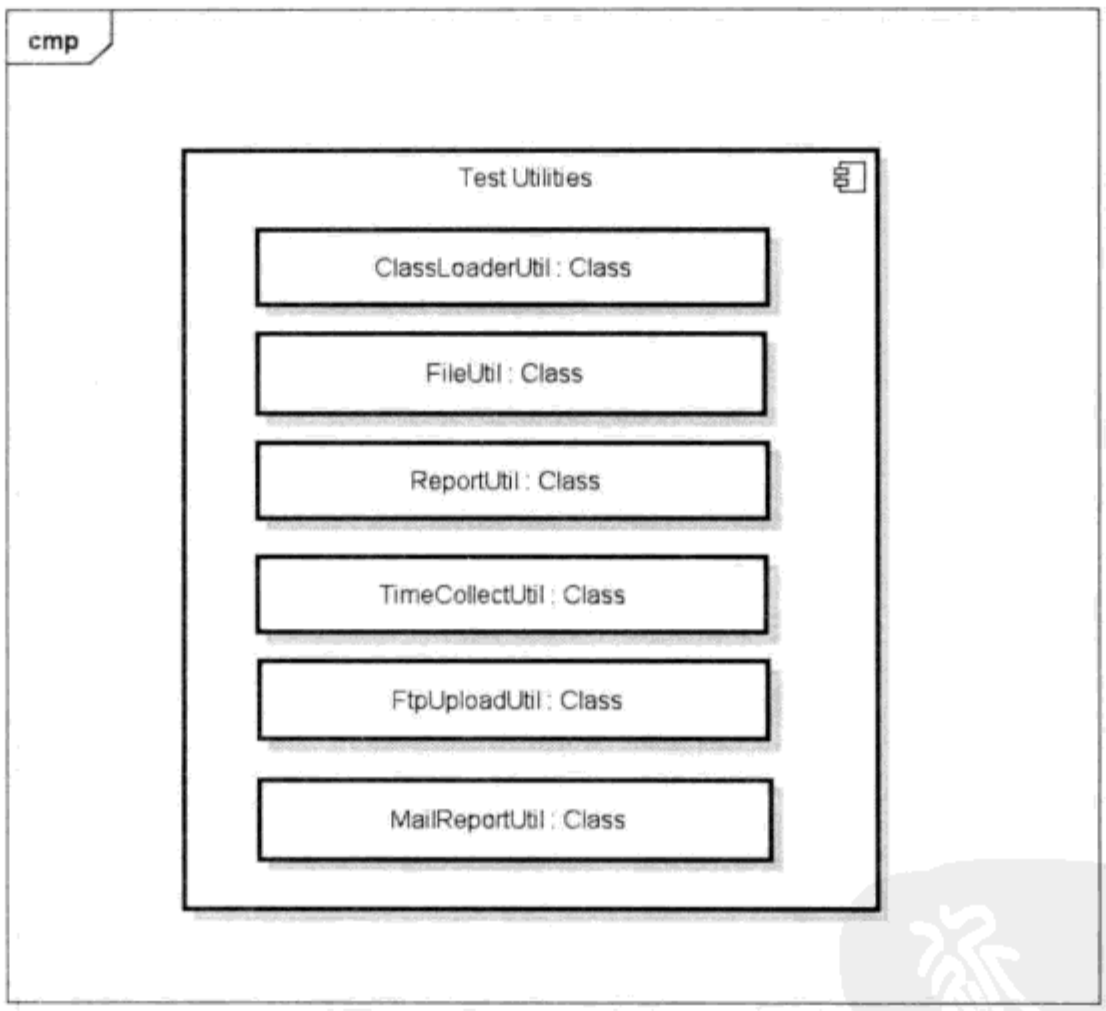


图 12-4　Test Utilities 组件图

TestUtilities 包含如下组件。

- **ClassLoaderUtil**：动态加载 Java Class 环境
- **FileUtil**：操作文件功能库，提供文件的读取、比对等常用操作
- **ReportUtil**：性能测试报告产生库，提供报告初始化，构建的 API

- **TimeCollectUtil：**时间度量数据的采集，负责后台事务时间的度量与采集工作
- **FtpUploadUtil：** 性能测试报告 FTP 上传功能
- **MailReportUtil：** 性能测试报告自动邮件通知功能

5. Test Observer（性能测试监控）

使用 Java 的观察者设计模式（Observer）可以实现对测试任务状态监控及数据采集，如图 12-5 所示。

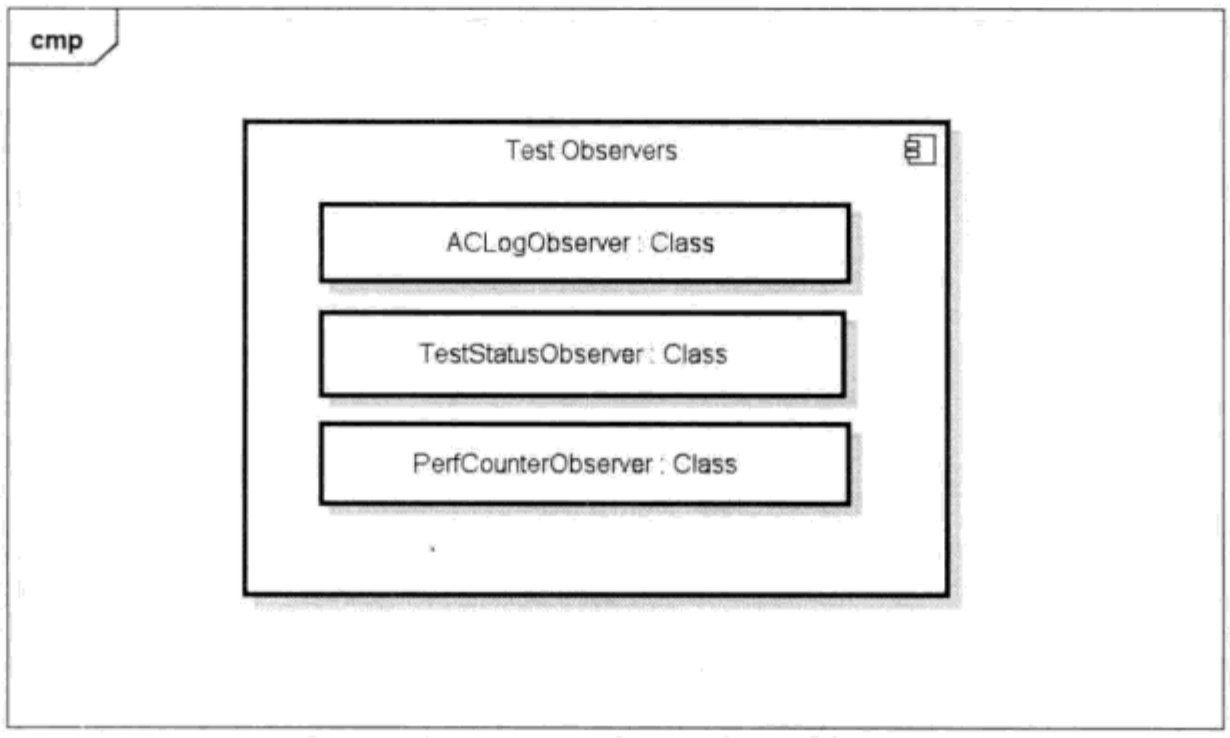


图 12-5 Test Observer 组件图

Test Observers 包含如下组件。

- **ACLogObserver：**采集工具运行日志
- **TestStatusObserver：**采集测试任务状态
- **PerfCounterObserver：**采集相关性能计数器

6. Test Report（性能测试报告）

使用 Java 的观察者设计模式（Observer）可以实现对测试任务状态监控及数

据采集，如图 12-6 所示。

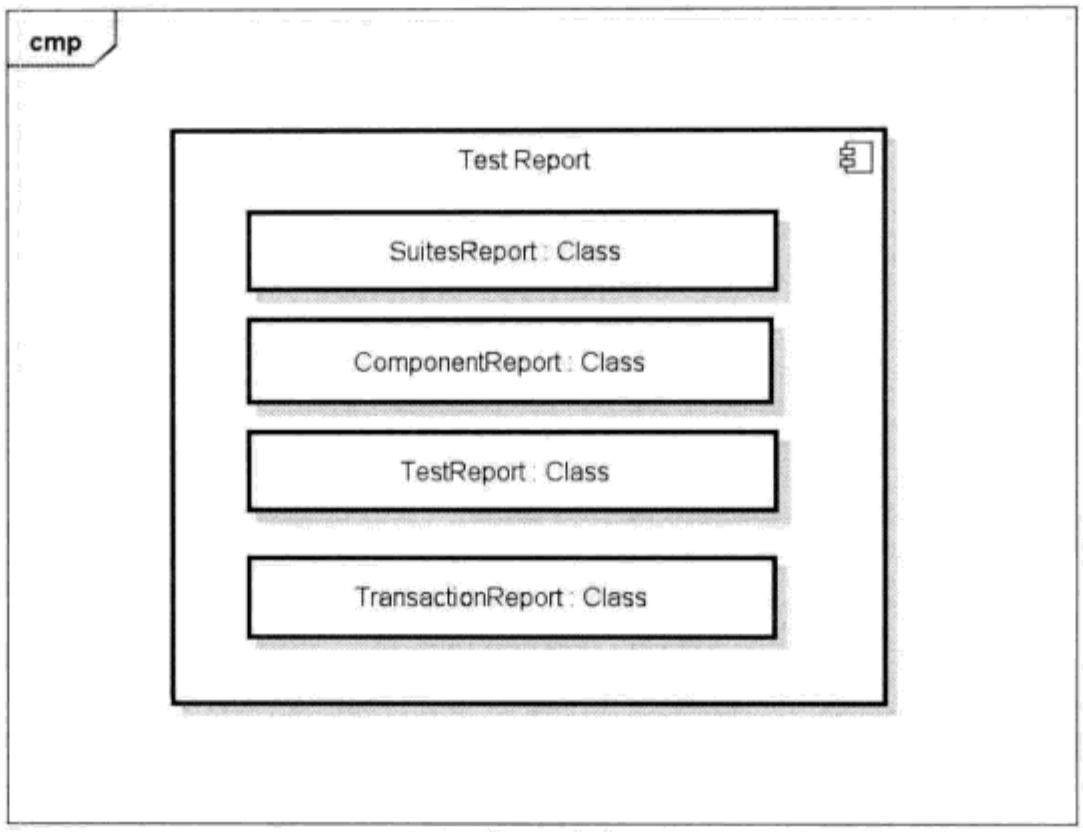


图 12-6　Test Report 组件图

Test Report 包含如下组件。

- **SuiteReport**：系统性能测试报告
- **ComponentReport**：组件性能测试报告
- **TestReport**：案例性能测试报告
- **TransactionReport**：事务性能测试报告

12.1.3　定义 XML 格式的性能测试任务文件

以测试任务为核心并驱动的性能测试会更加真实地模拟真实世界的业务行为。因为每个测试任务对象可以植入个性化的信息，比如随机时间种子、数据生成码点等（code point）。以下是一个用来定义测试任务的 TestJobFile.xml 文件：

```
<TestJob name="load test email service" description="load test " >
    <PropFile name="$ENV" location="instanceInfo.xml"/>
```

```
        <ConfFile name="$CONFIG" location="config.xml"/>
        <WebService name="SendEmail" description="Send email use
Soap request" mode="loadtest" threads="100" >
            <JobOutput name="$MAIL_SUBJECT"/>
            <Testdata type="xml" location="testdata\emailData.xml"/>
             <session>
                  <WSHOST value="ws.cesoo.info "/>
                  <WSPORT value="7777"/>
                  <WSUSER value="dev"/>
                  <WSPWD value="1234"/>
             </session>
             <SOAPRequest>
                   <URL value="URL=http://mail.cesoo.info/ws/message"/>
                   <SOAPEnvelope="<?xml version=\"1.0\"
                           encoding=\"utf-8\"?>
                           <soap:Envelope
                           xmlns:soap=\"http://schemas.xmlsoap.org/
                           soap/envelope/\"
                           xmlns:soapenc=\"
                           http://schemas.xmlsoap.org/soap/encoding/\"
                           xmlns:tns=\"urn:Cesoo\"
                           xmlns:types=\"urn:Cesoo/encodedTypes\"
                           xmlns:xsi=\"
                           http://www.w3.org/2001/XMLSchema-instance\"
                           xmlns:xsd=\"
                           http://www.w3.org/2001/XMLSchema\">
                          <soap:Body soap:encodingStyle=\"
                          http://schemas.xmlsoap.org/soap/encoding/\">
                          <tns:doCesooEmail>

                          <subject xsi:type=\"xsd:string\">
                          $DATAPOOL.SUBJECT</subject>
                          <to xsi:type=\"xsd:string\">
                          $DATAPOOL.TO</to>
                          <body xsi:type=\"xsd:string\">
                          $DATAPOOL.BODY</body>
                          <q xsi:type=\"xsd:string\">
                          LoadTesting</q>
                          <start xsi:type=\"xsd:int\">0</start>
                         <maxResults xsi:type=\"
                          xsd:int\">10</maxResults>
```

```
                    <filter xsi:type=\"xsd:boolean\">
                     false</filter>
                     <restrict xsi:type=\"xsd:string\" />
                    <safeSearch xsi:type=\"xsd:boolean\">
                     false</safeSearch>
                     <lr xsi:type=\"xsd:string\" />
                     <ie xsi:type=\"xsd:string\" />
                     <oe xsi:type=\"xsd:string\"/>
                     </tns:doCesooEmail>
                     </soap:Body>
                     </soap:Envelope>
                     "/>"
                     </SOAPRquest>
        </WebService>
    <IMAP name="checkEmail" depends="SendEmail">
        <JobInput name="$MAIL_SUBJECT"/>
        <CheckEmail>
            <CheckPoint>$MAIL_SUBJECT</CheckPoint>
        </CheckEmail>
    </IMAP>
    <AC name="EndJob" description=""  depends="">
        <BuildReport path="./"/>
            <UploadReport ToDo="Yes">
                <FtpServer>$CONFIG.FTP_SERVER</FtpServer>
                <FtpUser>$CONFIG.FTP_USER</FtpUser>
                <FtpPasswd></FtpPasswd>
                <FtpDir>$ENV.BUILD</FtpDir>
            </UploadReport>
            <MailReport ToDo="yes">
                <To>sheng.liu@live.cn</To>
                <Cc>sunshinelius@gmail.com</Cc>
                <Subject>This is report generated by AC</Subject>
            </MailReport>
        </AC>
</TestJob>
```

字段解释如下。

- **<PropFile>**：instanceInfo.xml 存储测试环境信息，如测试服务器、测试版本、账户、口令等。

- <ConFile>：config.xml 存储性能框架的配置信息，如 Ftp 服务器、邮件服务器等信息。
- Mode：有 FuncTest 和 LoadTest 两个选项。FuncTest 为默认的单线程功能测试，而 LoadTest 为多线程并发的负载测试。
- Threads：多线程并发数。
- <TestData>：性能测试数据池，支持 XML 和 XSL 两种。
- <WebService>：调用 Web Service 接口发送邮件。
- <JobOutput>：在发送邮件后，将 MAIL_SUBJECT 作为参数输出。
- $DATAPOOL：从 TestData 中读取的测试数据，格式为哈希表。
- <IMAP>：调用 IMAP 协议接口接收邮件，并进行验证。
- <JobInput>：从 AC 中获得 MAIL_SUBJECT 参数，并根据其进行验证。
- <AC>： Automation Center 的后台任务。
- <BuildReport>：构建性能测试报告。
- <UploadReport>：FTP 上传性能测试报告到服务器上。
- <MailReport>：邮件发送性能测试报告通知。

以上 TestJob 文件描述了一个性能测试的测试过程流程，各个 TestJob 的 depends 定义揭示了 AC 的运行过程。

AC 会首先运行名为 sendEmail 的 Web Service 负载测试，sendEmail 测试任务被克隆并发 100，调用服务器的 Web Service 收邮件接口，发送 soap 数据包，数据包里的邮件 subject，body，to 等字段由 AC 的 DATAPOOL 提供。发送完毕后，邮件 subject 被输出（JobOutput）。

SendEmail 成功后，IMAP 类型的 CheckMail 的测试任务开始运行，与服务器建立 IMAP 协议连接，从服务器接收邮件并根据输入（JobInput）对邮件标题进行验证。

可以看出，在 AC 的帮助下，性能测试模块的结构性，开发和运行效率都有大幅提高。

AC 不依赖于任何一种测试工具或脚本语言，可以将两个完全不同类型的接口协议整合在一起，成为一个完整的性能测试方案。比如，基于 SOAP 实现的 Web Service 和 IMAP 协议。

Depends 机制提高了自动化测试的执行效率，如果先决条件 TestJob 运行失败，则后续 TestJob 不会被运行。而 JobInput 和 JobOutput 为 TestJob 之间的数据交互提供了通道。

12.1.4　Automation Center 性能测试调度运行

Automation Center 的用户执行场景如图 12-7 所示。

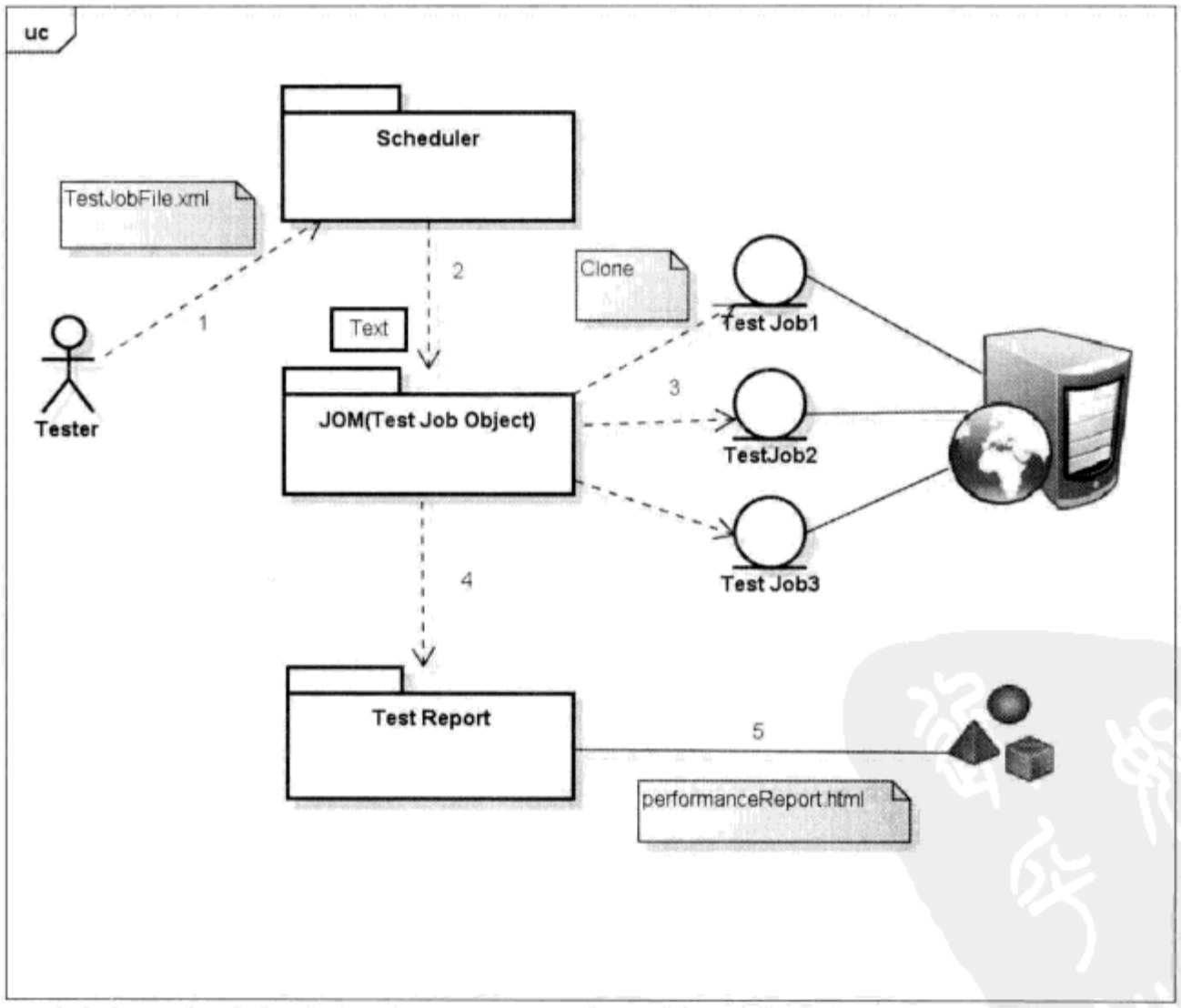


图 12-7　Automation Center 用户案例图

步骤如下：

（1）用户编辑 TestJobFile.xml，定义测试任务及场景。

（2）Scheduler 加载任务文件，执行测试任务，调度 AC 核心测试任务模型。

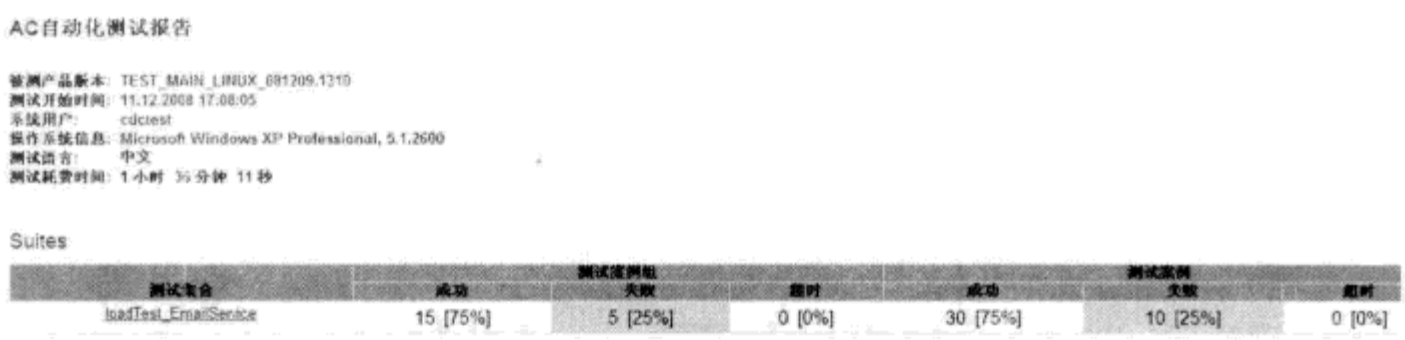
（3）测试任务通过 Java Clone 技术复制出多个子 TestJob，并发运行，向被测服务器施加负载压力。

（4）测试任务模型（JOM）运行结束后，各个子 TestJob 采集回度量数据，转化成测试报告对象模型（Test Report）。

（5）测试报告对象模型转化成基于 HTML 的 Web 页面。

12.1.5 性能测试报告诊断系统

1. 性能测试总览报告


AC自动化测试报告

被测产品版本: TEST_MAIN_LINUX_081209.1210
测试开始时间: 11.12.2008 17:08:05
系统用户: cdctest
操作系统信息: Microsoft Windows XP Professional, 5.1.2600
测试语言: 中文
测试耗费时间: 1 小时 35 分钟 11 秒

Suites

测试集合	测试逻辑组			测试案例		
	成功	失败	超时	成功	失败	超时
loadTest_EmailService	15 [75%]	5 [25%]	0 [0%]	30 [75%]	10 [25%]	0 [0%]

图 12-8 性能测试概要报告

包含如下信息。

- **被测产品版本**：被测软件系统的版本，用来记录回归测试信息。
- **测试开始时间**：性能测试开始的时间，由 Automation Center 自动生成。
- **操作系统信息**：运行操作系统环境信息，包括型号、版本、补丁信息，由 Automation Center 采集。
- **测试执行时间**：性能测试执行时间。
- **成功**：达到性能测试预期指标。
- **失败**：没有达到预期指标，或在执行过程中出现错误。

- **超时：**测试任务在运行过程中超时。

2. 细分案例报告

被测产品: OCSCORE_MAIN_LINUX_081209.1310
测试开始时间: 11.12.2008 17:07:54
测试系统用户: cdctest
操作系统信息: Microsoft Windows XP Professional, 5.1.2600
测试语言: de
测试耗费时间: 1 hr 36 min 11 sec

测试报告　　back to

TestCases	Tests 通过	失败	超时
obio_manuallyconfigureprofilewithassociation	1 [100%]	0 [0%]	0 [0%]
obio_connectobioafterinstallfromexefile	1 [100%]	0 [0%]	0 [0%]
obio_sendreplyallforwardmessage	3 [100%]	0 [0%]	0 [0%]
obio_createupdatepermanentlydeletenotes	3 [100%]	0 [0%]	0 [0%]
obio_createdltshftdeletregappwithallfields	3 [100%]	0 [0%]	0 [0%]
obio_createupdatepermanentlydeletetask	1 [33%]	2 [67%]	0 [0%]

图 12-9　细分案例列表

性能测试事务一览列表。

测试案例: obio_createupdatepermanentlydeletetask

运行结果		Failed
被测模块		OUTLOOKINT
功能点		MS Outlook Integration - OUTLOOKINT (OBIO)
案例描述		Create/Update/Shift-delete regular tasks by specifying data in all available fields
执行花费时间		5 min 40 sec
案例名称		-CreateTaskWithAllFields - Failed
	time	1 分钟 19 秒
	description	CreateTaskWithAllFields
	Check Point	Passed:Successfully go to folder [Mein Arbeitsbereich - shining liu:Aufgaben] Passed:Successfully create [Task] item Passed:Successfully open a new task dialog Passed:Successfully type [1229012122 CreateTaskWithAllFields Subject] on object [1229012122 CreateTaskWithAllFields Subject] Passed:Successfully type [CreateTaskWithAllFields Notes] on object [CreateTaskWithAllFields Notes] Passed:Task Title and Details set successfully Passed:Successfully type [12.12.2008] on object [12.12.2008] Passed:Successfully type [14.12.2008] on object [14.12.2008] Passed:Set task dates successfully Passed:Successfully open [Task] properties dialog Passed:task importance and Sensitivity successfully set to [Hoch and Vertraulich] Passed:Successfully type [28] on object [28] Passed:Task Status and Percentage Complete set successfully Passed:Successfully switch to tab [Details] in Task window Passed:Successfully type [4] on object [4] Passed:Successfully type [6] on object [6] Passed:Set task complete date, total and actual work successfully Passed:Successfully switch to tab [Details] in Task window Passed:Successfully type [Oracle] on object [Oracle] Passed:Successfully type [15] on object [15] Passed:Successfully type [Finance] on object [Finance] Passed:Task Companies, Mileage and Billing Info set successfully Passed:Update task successfully Passed:Successfully access Aufgaben: Eigenschaften - [Verwaltung] tab
	Failure	Failed:Failed to reset folder because there are pending upload changes Failed:Failed to reset folder Failed:Test:CreateTaskWithAllFields:All attempts for Recovery failed

图 12-10　失败交易诊断报告

可以在报告中查看失败交易的详细信息，包括执行时间、执行功能点、检查点通过情况。

3. Log 诊断



图 12-11　查看交易事务执行 log

4. snapshot 诊断

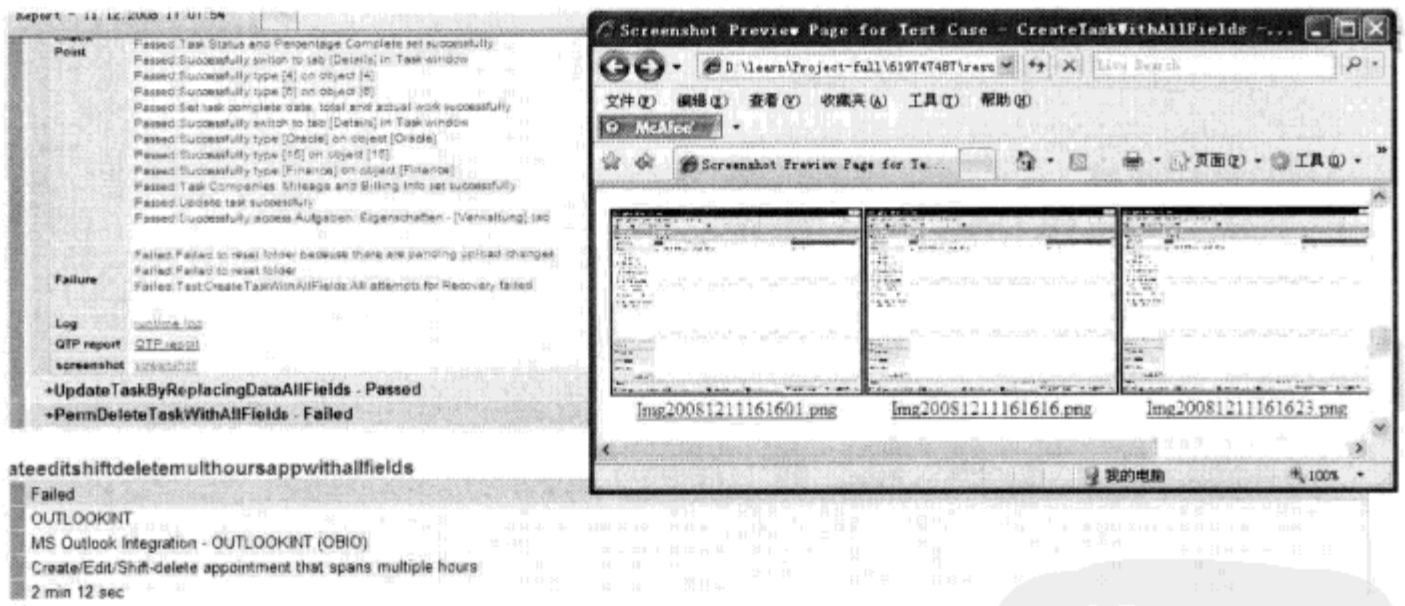


图 12-12　查看交易事务 snapshot

12.2　核心技术及具体实现方法

12.2.1　性能测试数据源格式定义

性能测试数据的特点是一个参数名可以对应多个值，从数据结构的角度来看，

就是一个 key（键名）对应多个 value（键值）。

结构如表 12-2 所示。

表 12-2　数据结构

Data key	Data Value 1	Data Value..	Data Value n
LoginName	张三		赵五
Password	123		abc
EmailSubject	Test Subject1		Test subject n
EmailBody	Test Body1		Test Body n

1. Excel 文件数据源

在 Excel 文件里，行代表参数名，列代表参数值，可形成参数矩阵，如图 12-13 所示。

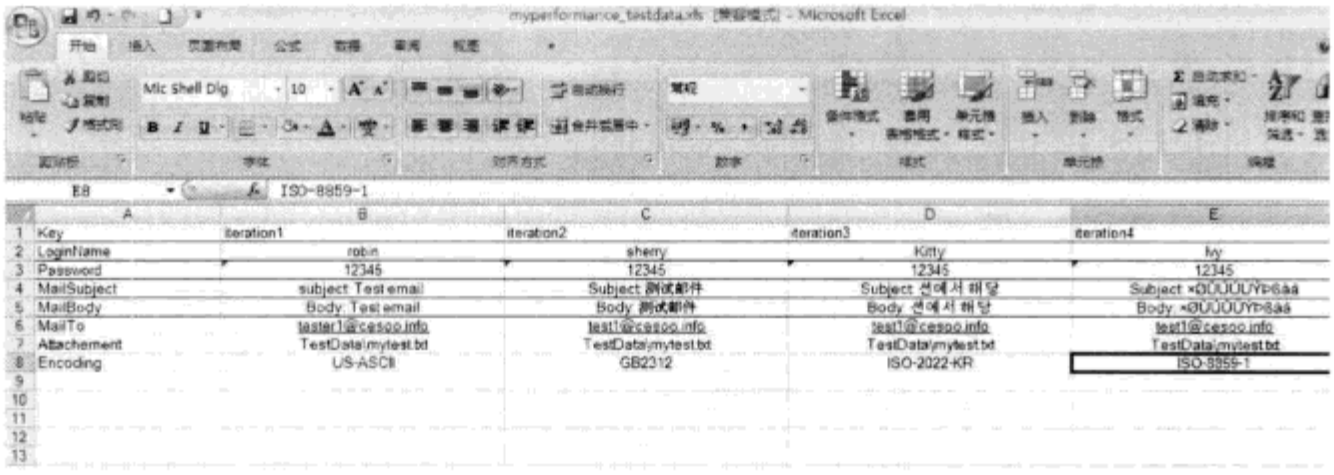


图 12-13　Excel 文件数据源

Java 通过 WorkBook 读取数据，如下所示：

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Properties;

import org.apache.poi.hssf.usermodel.*;
public class ReadExcel{
  public void loadExcelData(String filename){
```

```
try {
 FileInputStream fin = new FileInputStream(filename);
 wb = new HSSFWorkbook(fin);
 fin.close();
 sheet = wb.getSheetAt(0);
 HSSFRow row = sheet.getRow(1);
 int columns=row.getLastCellNum();
 for (short i=1; i<columns; i++)
 {
 try {
 HSSFCell cell = row.getCell(i);
 String columnname = cell.getStringCellValue();
 //System.out.println(" load header: "+columnname);
 header.put(columnname,new Integer(i));
  }
  catch (Exception e)
  {
    System.err.println(" Cell 0 "+i+" Failed to read");
    e.printStackTrace();
    return false;
  }
 }
}
        catch (Exception e)
        {
        e.printStackTrace();
        return false;
        }
}
}
```

2. XML 性能数据源

以 XML 文件方式存储性能测试数据，可以通过 node 嵌套来表现不同的参数名和参数值，如下所示：

```
<?xml version="1.0" encoding="UTF-8" ?>
<DataPool>
<Data indicator="en">
   <LoginName>robin</LoginName>
<Password> 12345</Password>
   <To>test1@cesoo.info</To>
   <MailBody> Body: Test email</MailBody>
```

```
    <MailSubject>subject: Test email </MailSubject>
<Encoding>US-ASCII</Encoding>
<Attach>TestData\mytest.txt</Attach>
</Data>
<Data indicator="ko">
    <LoginName>kitty</LoginName>
<Password> 12345</Password>
    <To>test1@cesoo.info</To>
    <MailBody>Body: 션에서 해당</MailBody>
    <MailSubject>Subject: 션에서 해당</MailSubject>
<Encoding>ISO-2022-KR</Encoding>
<Attach>TestData\mytest.txt</Attach>
</Data>
<Data indicator="zh_CN">
    <LoginName>sherry</LoginName>
<Password> 12345</Password>
    <To>test1@cesoo.info</To>
    <MailBody>Body: 测试邮件</MailBody>
    <MailSubject>Subject: 测试邮件</MailSubject>
<Encoding>GB2312</Encoding>
<Attach>TestData\mytest.txt</Attach>
</Data>
<Data indicator="Western">
    <LoginName>ivy</LoginName>
<Password> 12345</Password>
    <To>test1@cesoo.info</To>
    <MailBody>Body: xØÙÚÛÜÝÞßàá</MailBody>
    <MailSubject>Subject: xØÙÚÛÜÝÞßàá</MailSubject>
<Encoding>ISO-8859-1</Encoding>
<Attach>TestData\mytest.txt</Attach>
</Data>
</DataPool>
```

使用 Java XML DOM API 读取 XML 文件的示例如下：

```
Pubic class readXML{
    public void loadXMLFromFile(String xmlDataFile){
        NodeList rootNode=null;
        Element rootElement;
        //JOM jom = new JOM();
        try {
```

```
            DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
            DocumentBuilder builder;
builder= factory.newDocumentBuilder();
            Document document = builder.parse(xmlDataFile);
            rootElement = document.getDocumentElement();
            NodeList dataNodeList;
dataNodeList = rootElement.getElementsByTagName("DATA");
            int datasize = dataNodeList.getLength();
            for(int i=0;i<datasize;i++){
                Node datanode = dataNodeList.item(i);
                Element ele = (Element)dataNodeList.item(i);
                String indicator;
indicator = ele.getAttribute("Indicato");
                Properties dataprop = new Properties();
                NodeList paraNodeList;
                paraNodeList = datanode.getChildNodes();
                int parasize = paraNodeList.getLength();
                for(int j=0;j<parasize;j++){
                    Node paranode = paraNodeList.item(j);
                    if(paranode.getNodeType() == Node.TEXT_NODE)
                    continue;
                    String paraname = paranode.getNodeName();
                    String paravalue;
paravalue = paranode.getTextContent();
                    dataprop.setProperty(paraname,paravalue);
                }
                testDataHashArr.put(indicator,dataprop);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

12.2.2 测试程序与性能测试框架的接口实现

测试程序与性能框架对接采用对象继承的方法，示例代码如下。

```
Import framework.JobDOM.ACPerfJob;
public class MyPerfJob extends ACPerfJob{
```

```
        //Init the Performance Job
        public void Init(){
         //here will add some code to init the Job
        }
        //Run the performance Job
        public void Run(){
        //here will implement run of the job.
        }
        //End of the performance Job
        public void End(){
         //here could add some code to free or destroy the job
        }
}
```

MyPerfJob 类从 ACPerfJob 类继承，从而获得多种与框架母体通信的渠道，如图 12-3 所示。

表 12-3　函数

函数名	函数参数	接口描述
GetConfig(String confKey)	conKey- 配置项名	返回值 String 获得框架运行配置信息
GetEnv（String envKey）	EnvKey- 环境变量名	返回值 String 获得框架运行环境信息
GetData(String dataKey)	DataKey-测试数据参数名	返回值 String 从 DataPool 中获得测试数据



Socket 性能测试脚本：

```
Import framework.JobDOM.ACPerfJob;
import java.io.*;
import java.net.*;
public class MyPerfSocket extends ACPerfJob{
    private Socket Client;
    BufferedInputStream bis;
    BufferedOutputStream bos;
    public void Init(){
      String host = getEnv("HOST");
      Int port = getEnv("PORT");
      Client = new Socket(host,port);
    }
```

```
    public void Run(){
       bis = new BufferedInputStream(client.getInputStream());
       bos = new BufferedOutputStream(client.getOutputStream());
      String testData = getData("SOCKET_DATA");
       bos.write((testData).getBytes());
       bos.flush();
    }

    public void End(){
       client.close();
    }
}
```

Case **Socket 性能测试脚本：**

```
Import framework.JobDOM.ACPerfJob;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.Authenticator;
public class MyPerfSMTP extends ACPerfJob{
    Message msg;
    Session session;
    Transport trans;
    Properties props;
    public void Init(){
      String SMTP_HOST = getEnv("SMTP_HOST");
      int SMTP_PORT = getEnv("SMTP_PORT");
      props = new Properties();
      props.put("mail.smtp.socketFactory.port", SMTP_PORT);
      props.put("mail.smtp.host", SMTP_HOST);
      props.put("mail.smtp.auth", "true");
      props.put("mail.smtps.quitwait", "false");
      props.put("mail.smtp.port", SMTP_PORT);
      Authenticator auth = new SMTPAuthenticator();
     session = Session.getDefaultInstance(props, auth);
     msg = new MimeMessage(session);
```

```
    }

    public void Run(){
       String from = getData("MAIL_FROM");
      String to = getData("MAIL_TO");
       String subject = getData("MAIL_SUBJECT");
       String body = getData("MAIL_BODY");
       Msg. SetRecipients(to);
       Msg.setFrom(from);
       Msg.setSubject(subject);
       Msg.setBody(body);
       Transport.send(msg);
    }

    public void End(){

    }
}
```

12.2.3　测试任务克隆技术

每个测试任务（Test Job）在 Test Scheduler 的调度下，进行 clone，同时运行，向服务器发起多并发负载压力。所有的 Test Job 都继承于基类 Test JobElement，而 Test JobElement 实现了 clone 接口。

```
public class TestJobElement implements Cloneable{
    //.....
}
```

在 Test Scheduler 里，就可以实现对 Test Job 的 clone 了，如下：

```
for(int I=0;I<totalThreads;I++){
     TestJob newjob = (TestJob)JOM.deepCloneJob(testjob);
     Scheduler.addJob(newjob);
}
```

12.2.4 多任务通信技术

多 Test Job 通过父类 TestJobElement 进行通信和同步，如表 12-4 所示。

表 12-4 函数

函数名	函数参数	接口描述
Pass（String msg）	Msg-成功消息	向框架报出检查点成功的消息
Fail（String msg）	Msg-失败消息	向框架报出检查点失败的消息
Status（String status）	Msg-运行状态	向框架报出当前运行状态信息

父类通过轮询其下的子 Test Job 来获得运行信息。对超时的 Job 强制退出并销毁。

```
int iChilds = getChildJobsNum();
for(int i=0;i<iChilds;i++){
    TestJob child = getChildNodesByIndex(i);
    if(child.getElapsedTime()>TimeOut){
        String name = child.getName();
        log("job "+name " time out! Will be killed!")
        child.destroy();
     }
}
```

父类通过轮询其下的子 Test Job 来下达负载测试开始的命令。

```
int iChilds = getChildJobsNum();
  for(int i=0;i<iChilds;i++){
        TestJob child = getChildNodesByIndex(i);
        if(!child.IsRunning){
            child.Start();
         }
   }
```

父类通过轮询其下的子 Test Job 来下达负载测试停止的命令。

```
int iChilds = getChildJobsNum();
   for(int i=0;i<iChilds;i++){
          TestJob child = getChildNodesByIndex(i);
          if(child.IsRunning){
              child.Stop();
           }
    }
```

父类通过轮询其下的子 Test Job 来收集当前的性能监控信息。

```
int iChilds = getChildJobsNum();
   for(int i=0;i<iChilds;i++){
          TestJob child = getChildNodesByIndex(i);
          Metrics metric = child.getMetric();
           Scheduler.addMetric(metric)
    }
```

12.2.5　Observer 性能监控技术

Automation Center 通过 Java Observer 设计模式来实现对测试任务的性能监控及跟踪。

observer 模式定义对象间的一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。JDK 里提供的 observer 设计模式的实现由 java.util.Observable 类和 java.util.Observer 接口组成。从名字上可以清楚地看出两者在 Observer 设计模式中分别扮演的角色：Observer 是观察者角色，Observable 是被观察目标（subject）角色。

在 AC 里，测试任务（Test Job）作为被观察者（Observable），将会实时向观察者（Observer）通知各种信息。

我们定义三种观察者。分别记录框架日志、测试任务日志和测试任务性能度量信息。

1. 框架日志观察者

import java.util.Observer;

```
public abstract class FrameLogObserver implements Observer {
    public FrameLogObserver() {
    }
    //update 函数负责接收被观察者的信息
    public void update(Observable o, Object arg){
        if (arg instanceof String){
                //如果是 string 类型，将会写入 log4j 格式的框架日志文件
               Log4j.writeFrameLog(arg);
        }
    }
}
```

2. 测试任务日志观察者

import java.util.Observer;

```
public abstract class TestLogObserver implements Observer {
    public FrameLogObserver() {
    }
    //update 函数负责接收被观察者的信息
    public void update(Observable o, Object arg){
        if (arg instanceof String){
                //如果是 string 类型，将会写入 log4j 格式的测试任务日志文件
               Log4j.writeTestLog(arg);
        }
    }
}
```

3. 测试任务性能度量观察者

import java.util.Observer;

```
public abstract class TestMetricObserver implements Observer {
    public FrameLogObserver() {
    }
    //update 函数负责接收被观察者的信息
    public void update(Observable o, Object arg){
```

```
            if (arg instanceof Metric){
                    //如果是 Metric 对象，将会写入 Report 对象中的 metric 中，
供 report 分析
                Report.addMetric(arg);
            }
        }
    }
```

上面三种观察者在被观察的测试任务（TestJob）中注册后，就可以自动获得消息了。

4. 测试任务被观察者

import java.util.Observable;

```
public class TestJob extends Observable{
    public TestJob() {
    }
    //注册三种观察者
    public void subscribeObserver(){
        super.addObserver(FrameLogObserver);
        super.addObserver(TestLogObserver);
        super.addObserver(TestMetricObserver);
    }
    //通知 FrameLogObserver 写入框架日志
    public void outputFrameLog(String strLog){
        super.setChanged();
        notifyObservers(FRAME_LOG_PREFIX +strLog);
    }
    //通知 TestLogObserver 写入测试任务日志
    public void outputTestLog(String strLog){
        super.setChanged();
        notifyObservers(TEST_LOG_PREFIX+strLog);
    }
    //通知 TestMetricObserver 记录性能度量信息
    public void outputJobLog(Metric metric){
        super.setChanged();
        notifyObservers(metric);
    }

}
```

12.2.6 基于 XSL 技术的性能测试报告构建

性能测试报告是自动化测试一个非常重要的部分，因为：

（1）测试报告是自动化测试展现成果的窗口。

测试经理、研发总监、质量总监等角色都会经常关注性能测试报告的状态。一份形象直观、数据全面的测试报告往往会给领导留下深刻的印象，从而能够加大对性能测试的支持力度。

（2）测试报告对技术人员（性能测试人员和开发人员）来说，也是诊断性能测试运行异常或错误的重要线索。在软件开发过程中，通常要有一套严格定义，形式完备的日志系统来帮助开发人员追踪和分析问题。在自动化测试开发过程中，对报告、日志的要求更加严格，因为性能测试脚本失败的原因可能更加复杂，有可能是环境问题，或者被测软件的 bug，甚至测试脚本本身的问题。如何定位这些问题，必然对测试报告有更高的要求。

从以上两点分析，我们可以看出，性能测试报告有两面特征，在前端应该直观简洁，让人一目了然，同时在后端应该存储尽可能多的度量数据，供软件人员分析。

为了满足不同角色人员的需要，通常我们的做法是将性能测试报告分为两级：总览报告（overview report）和细分报告（breakdown report）。

在这里，我们采用 XML 和 XSL 技术来进行报告呈现。

XSL（eXtensible Style sheet Language），即可扩展样式表语言，是 XML 的亲密伙伴，在 XML 数据中指定相应的 XSL，即可把数据按照一定格式进行呈现。

Report 采用 XML 和 XSL 技术，做到数据与呈现方式分离，实现了一处数据，多样呈现的思想，如图 12-14 所示。

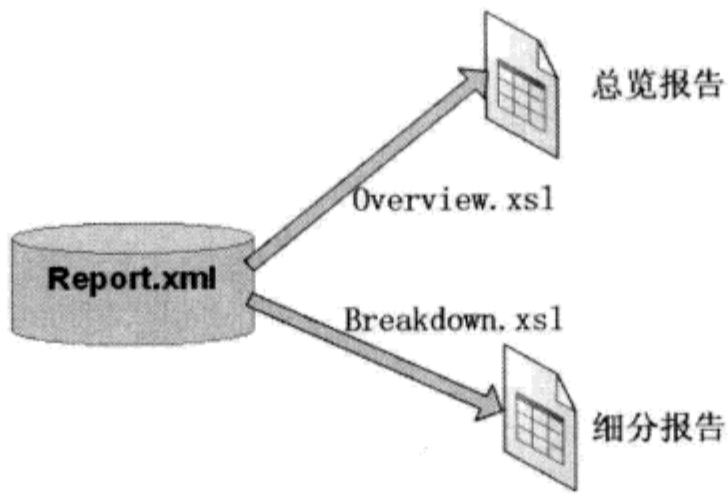
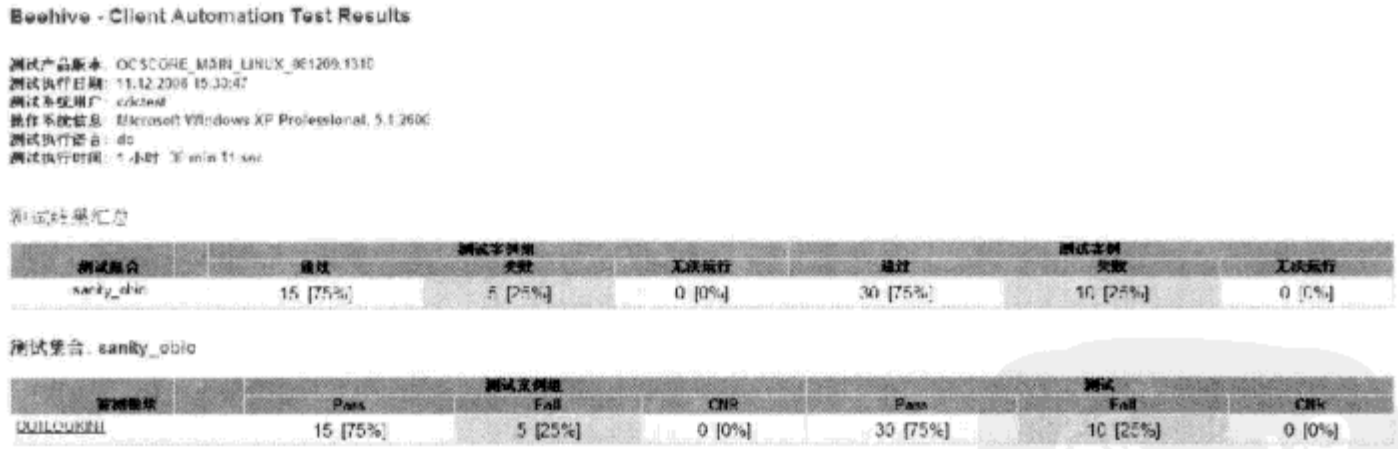


图 12-14　基于 XML 和 XSL 技术的测试报告

在 XML 文件 header 中指定相应的总览格式报告的 reporting-summary.xsl 处理文件：

```
<?xml version="1.0" encoding="gb2312"?>
<?xml-stylesheet type="text/xsl" href=" reporting-summary.xsl"?>
……
```

可获得总览统计报告，如图 12-15 所示。


Beehive - Client Automation Test Results

测试产品版本：OCSCORE_MAIN_LINUX_061209.1310
测试执行日期：11.12.2006 15:33:47
测试系统用户：cdctest
操作系统信息：Microsoft Windows XP Professional, 5.1.2600
测试执行平台：do
测试执行时间：1 小时 36 min 11 sec

测试结果汇总

测试集合	测试案例集 通过	测试案例集 失败	测试案例集 无法运行	测试案例 通过	测试案例 失败	测试案例 无法运行
sanity_ohio	15 [75%]	5 [25%]	0 [0%]	30 [75%]	10 [25%]	0 [0%]

测试集合：sanity_ohio

测试模块	测试案例组 Pass	测试案例组 Fail	测试案例组 CNR	测试 Pass	测试 Fail	测试 CNR
OUTLOOKEXE	15 [75%]	5 [25%]	0 [0%]	30 [75%]	10 [25%]	0 [0%]

图 12-15　基于 reporting-summary.xsl 的总览报告呈现

在相同的 XML 文件 header 中指定相应的总览格式报告的 reporting-detail.xsl 处理文件：

```
<?xml version="1.0" encoding="gb2312"?>
<?xml-stylesheet type="text/xsl" href=" reporting-detailxsl"?>
……
```

测试案例: obio_createupdatepermanentlydeletetask

运行结果	Failed
被测模块	OUTLOOKINT
功能点	MS Outlook Integration - OUTLOOKINT (OBIO)
案例描述	Create/Update/Shift-delete regular tasks by specifying data in all available fields
执行花费时间	5 min 40 sec
案例名称	+CreateTaskWithAllFields - Failed +UpdateTaskByReplacingDataAllFields - Passed -PermDeleteTaskWithAllFields - Failed

time	1分钟 9秒
description	PermDeleteTaskWithAllFields
Check Point	Passed:Successfully open Outlook Advanced Find Passed:Successfully select Advanced Find search type [Aufgaben] Passed:Successfully select folder [Mein Arbeitsbereich - shining liu\Aufgaben] Passed:Successfully type [1229012201 EditTaskWithAllFields Subject] on object [1229012201 EditTaskWithAllFields Subject] Passed:Successfully enter search string [1229012201 EditTaskWithAllFields Subject] and select search field [Nur im Feld "Betreff"] Passed:Successfully select [0] item in window [Erweiterte Suche] Passed:Succeeded to delete #0 item permanently Passed:Successfully access Aufgaben: Eigenschaften - [Verwaltung] tab
Failure	Failed:Failed to reset folder because there are pending upload changes Failed:Failed to reset folder Failed:Test:PermDeleteTaskWithAllFields:All attempts for Recovery failed.
Log	runtime.log
QTP report	QTP report
screenshot	screenshot

图 12-16 基于 reporting-detail.xsl 的细分报告呈现

从图 12-16 可以看出，在测试报告中采用标准的 XML 和 XSL 格式定义。可以大大提高性能测试框架的可扩展性，测试人员完全能够根据企业的实际环境，做出基于框架的二次开发，比如多种报告的整合、测试报告自动入库等，这无疑会增加软件性能测试的价值含量。

12.3 总结

性能测试框架的设计和开发要考虑到多线程同步，以及度量数据采集等关键技术问题，因此一个高质量的设计方案，不仅能够解决这些技术难点，而且如果企业有长期性能需求的话，还应该考虑有良好的可扩展性和可调式性。

有关更多性能框架信息，可以参看作者的个人博客 http://www.cesoo.info。



附录A

有关LoadRunner的常见问题解答

1．500 个用户的 License，能否做 10000 个用户的并发测试？

答：10000 个用户的并发，这是一个非常大的数目，只有巨型的软件系统才会有这样的性能需求。比如面向全球用户的 gmail 邮件系统，为了实现这样庞大的性能需求，除了软件程序的性能要到达一定指标，其后台的服务器集群架构及部署是非常复杂的，可以说，gmail 这样的系统一旦开始运转，其背后不是一台服务器在战斗，而是一个庞大的集群及架构部署。

而一般的商业软件系统则不会有这么高的硬件成本，因此，笔者个人建议提这个问题的朋友认真考虑一下你的软件系统是否真的有 10000 个用户并发的性能需求。为了搞清楚它，你需要逐一澄清以下几个事实：

（1）并发指的是多个客户端同时对服务器发起网络链接请求，并发送数据。

从服务器看来，它必然要开启大量的线程，消耗内存、网络套接字、句柄等

资源。这是并发测试的考察目标所在。

（2）并发用户数！=系统注册用户数。

显然，不是所有的系统注册用户都是活跃用户（经常登录系统的用户），而且，不是所有的活跃用户都能构成并发关系。因此，软件系统性能需求的并发用户数要远远小于系统注册用户数。具体占多少比例，可依据不同的业务模型而定。

（3）并发用户数！=在线用户。

“用户在线”看起来像是系统级别的概念，这是它能迷惑很多性能测试工程师的原因，误以为“用户在线”是一个标准的性能指标，能够衡量不同的软件系统。事实往往是拨云见山，其实“用户在线”是一个软件应用层次上的概念。也就是说，它在不同的软件系统中有不同的实现，甚至千差万别。

我们经常说 QQ 某天有多少用户在线，也经常说某网站曾经创下最多在线用户的记录。但如果我们从系统的角度来看两者的在线，就会发现两者在不在一个级别上：

- QQ 在线主要靠心跳报文来维持在线，即每隔一定时间就在 Socket 连接上和 Server 交互数据。Server 由此判断客户端还“活着”（在线），这也是心跳报文的命名原因。有兴趣的朋友可以用 LoadRunner 的 Winsock Vuser 录制 QQ 活动，你会发现，即使你没有通过 QQ 发消息，而脚本仍会记录下一些 Socket 交互，而且是有规律的，这就是用来维持在线的心跳报文。
- 而网站在线目前很多实现方法都是靠 Session 来维持用户在线的，如果 Session 存在 Cookie 里，那么在线就是非常简单的事情，几乎不占用什么 CPU 资源。因为 Cookie 只会占用硬盘空间或很少的内存空间。服务器通过校验客户端发送的 Session 是否超时来判断它是一个“活着”的用户，还是新用户。因此我们常看到某某著名网站论坛在线用户能创下几万甚至几十万的记录。

而并发用户是要真刀真枪地干，要让服务器乖乖干活的，因此和在线用户绝不可同日而语。

通过以上问题的澄清，如果你还确认性能需求还是并发用户 10000 个的话，那么恭喜你可以向老板申请增加性能测试的开支了。然后考虑通过下面两种方式完成这个盛大的任务：

（1）通过 LoadRunner 来完成，是否能够使用这种方式取决于被测系统的架构。LoadRunner 的 License 是基于虚拟用户的，而虚拟用户可以以进程方式运行，那么我们知道进程里可以开启多个线程。OK，剩下的你自己去考虑吧。

（2）无论如何，要让 LoadRunner 实现 10000 个用户并发不是一件很容易的事情（即使你有了 10000 个用户的 License），LoadRunner 胜任负载测试，压力测试却不是它的长项。因此在这种情况下，可以考虑自己写程序实现压力测试。

2．听说 LoadRunner 可以调用 QTP 脚本，那和 Web Click Vuser 有什么区别？

答：QTP 脚本的确可以被 LoadRunner 调用，但如果你据此认为 LoadRunner 可以使用 QTP 脚本做性能测试，那就错了。因为使用过 QTP 的朋友知道，QTP 脚本是基于界面事件的，也就是说，QTP 脚本在运行时，它要独占一台计算机，因为键盘和鼠标资源都是唯一的。因此，如果把 QTP 脚本引用到 LoadRunner 里来做负载测试，那就必须是一个 Load Generator 只能开启一个虚拟用户！这意味着如果你要做 100 个用户的并发，那就需要 100 台安装 Load Generator 的计算机。因此，使用 QTP 脚本做负载测试理论上可行，但实践上却行不通。这是和 Web Click Vuser 最大的区别之处。

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

Broadview®
www.broadview.com.cn

《性能测试从零开始——LoadRunner 入门与提升》

读者交流区

尊敬的读者：

感谢您选择我们出版的图书，您的支持与信任是我们持续上升的动力。为了使您能通过本书更透彻地了解相关领域，更深入的学习相关技术，我们将特别为您提供一系列后续的服务，包括：

1. 提供本书的修订和升级内容、相关配套资料；
2. 本书作者的见面会信息或网络视频的沟通活动；
3. 相关领域的培训优惠等。

请您抽出宝贵的时间将您的个人信息和需求反馈给我们，以便我们及时与您取得联系。

您可以任意选择以下三种方式之一与我们联系，我们都将记录和保存您的信息，并给您提供不定期的信息反馈。

1．电子邮件

您可以发邮件至jsj@phei.com.cn或editor@broadview.com.cn。

2．信件

您可以写信至如下地址：北京万寿路173信箱博文视点，邮编：100036。

3．读者电话

您可以直接拨打我们的读者服务电话：010-88254369。

在您选择的联系方式中，您还可以告诉我们更多有关您个人的情况，及您对本书的意见、评论等，内容可以包括：

（1）您的姓名、职业、您关注的领域、您的电话、E-mail地址或通信地址；

（2）您了解新书信息的途径、影响您购买图书的因素；

（3）您对本书的意见、您读过的同领域的图书、您还希望增加的图书、您希望参加的培训等。

如果您在后期想退出读者俱乐部，停止接收后续资讯，只需编写邮件“退订+需退订的邮箱地址”发送至邮箱：market@broadview.com.cn 即可取消该项服务。

同时，我们非常欢迎您为本书撰写书评，将您的切身感受变成文字与广大书友共享。我们将挑选特别优秀的作品转载在我们的网站（www.broadview.com.cn）上，或推荐至CSDN.NET等专业网站上发表，被发表的书评的作者将获得价值50元的博文视点图书奖励。

我们期待您的消息！

博文视点愿与所有爱书的人一起，共同学习，共同进步！

通信地址：北京万寿路 173 信箱　博文视点（100036）　　电话：010-51260888

E-mail：jsj@phei.com.cn，editor@broadview.com.cn

www.phei.com.cn
www.broadview.com.cn

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Broadview®
WWW.BROADVIEW.COM.CN

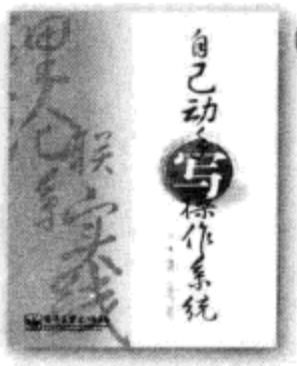
Csdn.net

技术凝聚实力 专业创新出版

博文视点（www.broadview.com.cn）资讯有限公司是电子工业出版社、CSDN.NET、《程序员》杂志联合打造的专业出版平台，博文视点致力于——IT专业图书出版，为IT专业人士提供真正专业、经典的好书。

请访问 www.dearbook.com.cn（第二书店）购买优惠价格的博文视点经典图书。
请访问 www.broadview.com.cn（博文视点的服务平台）了解更多更全面的出版信息；您的投稿信息在这里将会得到迅速的反馈。

博文本版精品汇聚



自己动手写操作系统

用理论指导动手实践，用实践深化理解理论。
出版一年累计销量20000册，广受读者好评！
于渊 编著　尤晋元 审校
ISBN 7-121-01577-3　48.00元（含光盘1张）



.NET 2.0面向对象编程揭秘

从跨语言角度全面介绍C#2.0语言特性
展示VS2005集成开发环境实用技巧
系统剖析 .NET核心技术内幕
深入探讨CLR内部运行机理
金旭亮 著
ISBN 978-7-121-03975-1　69.8元（含光盘1张）



游戏之旅——我的编程感悟

历时一载写就，凝结十年心得。
《仙剑奇侠传》之父姚壮宪热情推荐，技术作家孟岩高度评价！
云风 著
ISBN 7-121-01609-5　46.00元



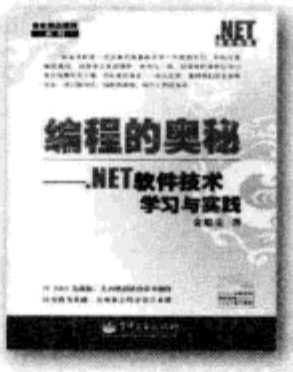
SOA原理.方法.实践

IBM资深架构师—毛新生主编
IBM中国开发中心技术经典呈现！
毛新生　主编
ISBN 978-7-121-04264-5　49.80元



Windows环境下32位汇编语言程序设计（第2版）

畅销书升级版！
罗云彬 编著
ISBN 7-121-02260-5　79.00元（含光盘1张）



编程的奥秘 —.NET软件技术学习与实践

步入编程之门，领略软件之美！
以.NET为载体，全面把握软件技术脉络！以实践为基础，直观体会程序设计本质！
金旭亮 著
ISBN 7-121-01820-9　65.00元（含光盘1张）



Ruby on Rails快速Web应用开发实战

国内第一本原创RoR图书。
实例引导，学以致用！
柳靖、曹璐、赵丹 编著
ISBN 7-121-02913-8　39.00元



Ajax开发精要 —概念、案例与框架

国内第一本重量级原创Ajax图书，由一线开发人员编写，上市以来持续热销！
柯自聪 编著
ISBN 7-121-02468-3　45.00元

博文视点资讯有限公司
电　　话：（010）51260888　传真：（010）51260888-802
E-mail：market@broadview.com.cn（市场）
　　editor@broadview.com.cn　jsj@phei.com.cn（投稿）
通信地址：北京市万寿路173信箱 北京博文视点资讯有限公司
邮　　编：100036

电子工业出版社发行部
发 行 部：（010）88254055
门 市 部：（010）68279077　68211478
传　　真：（010）88254050　88254060
通信地址：北京市万寿路173信箱
邮　　编：100036

博 文 视 点 · I T 出 版 旗 舰 品 牌

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：（010）88254396；（010）88258888

传　　真：（010）88254397

E-mail：　dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

　　电子工业出版社总编办公室

邮　　编：100036